EICHLER · GROHMANN

ATARI®

600 XL / 800 XL

INTERN

EIN DATA BECKER BUCH

EICHLER · GROHMANN

ATARI®

600 XL / 800 XL

INTERN

EIN DATA BECKER BUCH

ISBN 3-89011-053-3

Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technische Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Wichtiger Hinweis:

Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke gedacht und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

INHALTSVERZEICHNIS

SEITE

E I N L E I T U N G

Lieber Leser,

gleich zu Anfang dieser Einleitung möchten wir uns bei Ihnen bedanken, daß Sie dieses Kapitel überhaupt lesen. Hoffentlich blättern Sie nicht jetzt gleich um. Dies wäre kurzsichtig gehandelt, wie die Autoren aus eigener Erfahrung berichten können.

Sicherlich haben Sie wie wir nach dem Kauf Ihres Atari, sofern Sie einen besitzen, zuerst das Gerät ausprobiert, bevor Sie die Bedienungsanleitung gelesen haben. Bei genauerem Studium der mitgelieferten Dokumentation werden Sie dann sicherlich auch festgestellt haben, daß die interessantesten Angaben fehlen.

Dies ist in der Computerbranche leider üblich. Zudem gibt es über das Atari-System eine Reihe von Einzelpublikationen, die jeweils nur ein Spezialgebiet ohne Betrachtung anderer und oft auch unvollständig oder gar falsch erfassen. Bitte denken Sie nicht, daß wir für unser Buch Vollständigkeit oder gar Fehlerfreiheit beanspruchen; wir versuchen lediglich, ein Buch zu schreiben, das möglichst umfassend die Atari - Hardware und das Betriebssystem erläutert und allein schon dadurch die vielfältigen Möglichkeiten des Atari aufzeigt. Zudem werden wir an Stellen, die uns problematisch erscheinen, Beispiele einfügen.

Dieses Buch ist also besonders für den gedacht, der das Prinzip des Computers und das Basic bereits ausreichend

beherrscht, vielleicht auch schon weiß, was hexadezimale Zahlen sind. Im nächsten Kapitel erklären wir auch noch kurz den Umgang mit dem uns normalerweise leider fremden Zahlensystem.

Wenn man ein Buch über Computer schreibt oder auch einfach nur über sie spricht, so stellt sich immer die Frage nach der Wortwahl. Da in der gesamten Informatik Englisch Fachsprache ist, sind praktisch alle Fachausdrücke ebenfalls Englisch. Man muß sich also darüber klar werden, welche Ausdrücke man eindeutscht und für welche man den englischen Ausdruck beibehält.

Bereits in der Branche akzeptierte Fachausdrücke sollte man unbehandelt lassen, allerdings nur, sofern es nicht bereits einen ebenso akzeptierten deutschen Ausdruck dafür gibt. Dies sichert nach einiger Eingewöhnungszeit leichtere Verständigung mit anderen. Bei der Übersetzung oder beim "Eindeutschen" sollte man darauf achten, wenigstens "richtiges" Deutsch zu benutzen. Wer von "editieren" redet, weiß nicht, daß die Übersetzung schon in Form von "edieren" vorhanden ist. Ungläubige mögen den Duden zu Rate ziehen.

Weiter ist unbedingt zu beachten, daß keine sogenannten "Sekretärinnenübersetzungen" entstehen. Zum Beispiel ist es unsinnig und sogar falsch, den Ausdruck "Symbol table" in "Zeichentisch" zu übersetzen, richtig wäre "Symboltabelle". Allgemein sollte man immer versuchen, ein gesundes Mittelmaß zu finden. So ist es unsinnig, den Fachausdruck "Editor" korrekt in "Textbearbeiter" zu übersetzen. Wer jedoch bei Leitungen von "Lines" spricht, täuscht im allgemeinen Fachwissen vor und sollte weiter nur versuchen, Gäste von Stehparties zu beeindrucken und sich nicht wundern, wenn ihn wirkliche Fachleute belächeln.

Vielleicht kann man aber auch durch eine günstig gewählte Übersetzung das englischsprachige Original an Klarheit übertreffen. Wir werden zum Beispiel statt des in Deutschland durchaus üblichen Ausdrucks "Display-List" den Begriff "ANTIC-Programm" verwenden.

Ein weiteres Problem ergibt sich bei der Verwendung der Artikel bei englisch belassenen Ausdrücken. Soll man den Artikel übersetzen ? Wir glauben, daß es dafür keine allgemeingültige Regel gibt. Allerdings hat sich für die meisten Wörter eine Form besonders durch den Univeritätsbetrieb eingebürgert. Wir versuchen, diese, soweit sie uns bekannt ist, zu verwenden.

In der Öffentlichkeit hat Ataris Ruf leider etwas gelitten. Dies liegt vor allem daran, daß Atari eine Zeitlang durch konsequentes Nichteinhalten jeglicher Liefertermine glänzte. Durch die in dieser Zeit sehr hohen Verkaufszahlen der Konkurrenz, im Speziellen Commodore C-64, erhielten die Atari-Geräte den Ruf, auch technisch grundsätzlich schlechter zu sein.

Wir meinen dagegen, mit der Ansicht, beide Geräte auf eine Stufe zu stellen, der Wahrheit wesentlich näher zu kommen. Natürlich könnte man anführen, daß der C-64 eine wesentlich einfachere Sprite-Verarbeitung hat, das heißt, daß sich sehr leicht bewegliche Spielfiguren darstellen und verarbeiten lassen. Für den versierten Atari-Programmierer bieten sich dagegen mit der Player-Missile-Grafik zwar nicht so leicht zu programmierende, aber wesentlich effektvollere Bildaufbauten. Mit entsprechenden Tricks lassen sich auch erheblich mehr Objekte, als "in der Grundausstattung" vorhanden, darstellen.

Den größten Leistungsvorsprung findet man beim Atari im ANTIC-Baustein. Dieser Baustein stellt einen zweiten Prozessor dar, der auch über einen eigenen Maschinencode verfügt. Die einzelnen Befehle des ANTIC erzeugen bei ihrer Ausführung jeweils Teile des Gesamtbildes. Dabei ist es möglich, verschiedene Grafikbetriebsarten, sogenannte Grafikmodes, auf einem Bild frei zu kombinieren. Zum Beispiel kann man diverse Textdarstellungen mit unterschiedlich hoch auflösenden Blockgrafikmodes auf einem Bild kombinieren. Dieses Konzept findet man auch bei erheblich teureren Systemen nur sehr selten.

Die Tonerzeugung auf den vier grundsätzlich unabhängigen Tonkanälen ist ebenfalls sehr sauber entwickelt. Sie kann sich nach unserer Meinung mit allen anderen auf dem Homecomputermarkt befindlichen Systemen messen.

Das Basic ist aufgrund seiner eingeschränkten Größe in der Verarbeitung von Strings (Zeichenketten) unglücklich und gewöhnungsbedürftig, liefert aber gerade für den Einsteiger vielfältige Möglichkeiten durch zahlreiche schnelle Grafik- und Tonbefehle. Es ist grundsätzlich leider nicht üblich, mit dem bereits eingebauten Basic die Hardware so leicht und effizient auszunutzen, wie es erfreulicherweise bei Atari der Fall ist. Gerade für den Anfänger ist es angenehm, wenn er häufig benötigte Befehle nicht erst in teuer zu erstehenden oder illegal (schwarz) zu kopierenden Erweiterungen findet.

Dieses Buch soll sowohl "durchlesbares" Lehrbuch als auch Nachschlagewerk sein. Wir haben daher sowohl ein Register der verwendeten LABELS, als auch ein INHALTSREGISTER im Buch aufgenommen. Dabei haben wir beide alphabetisch geordnet. Das Labelregister verweist auf die hexadezimalen Adressen, die das jeweilige Label repräsentiert. Das Kapitel über das Betriebssystem ist grundsätzlich nach Adressen geordnet. Das Inhaltsregister verweist dagegen auf Seitenzahlen.

Hexadezimale Zahlenwerte werden wir immer durch ein vorangestelltes "$"-Zeichen kennzeichnen. Binäre Werte erhalten an Stellen, die Unklarheiten mit sich bringen können, ein vorangestelltes "%"-Zeichen.

DAS PRINZIP EINES MIKROCOMPUTERS

1) ALLGEMEINES

2) VON BITS UND BYTES

3) DIE HARDWARE EINES MIKROCOMPUTERS

4) HIGH UND LOW

"Computer können uns nicht einen einzigen Hintergedanken abnehmen" sagte einmal eine bedeutende amerikanische Firma in ihrer Werbung - und hat damit völlig Recht. Aber was können sie dann?

Die meisten Besitzer von Mikrocoputern wenden ihre Geräte auf die unterschiedlichsten Arten und Weisen an. Der größte Teil spielt ganz einfach damit oder, genauer, mit der darauf laufenden Software. Ein weiterer Teil benutzt den ATARI zum Programmieren oder zur Text- und Datenverarbeitung und ein, wenn auch nur kleiner, Teil benutzt ihn zur Realisierung von Steuerungsaufgaben.

Computer können lediglich das, was man ihnen beigebracht hat. Jede Regel für eine Entscheidung, bzw. jede Regel nach der andere Regeln aufgestellt werden sollen, den gesamten Arbeitsablauf, sowie das gesamte Wissen, das zur Lösung notwendig ist, muß man dem Computer in einer für ihn verständlichen Sprache mitteilen. Wenn man sich überlegt, was dies bedeutet, so kommt man zu dem Schluß, daß Computer an sich "ziemlich doof" sind.

Wissenschaftler, die sich mit der INTELLIGENZ verschiedener Lebewesen und auch des Mikrocomputers befaßt haben, kamen zu dem Schluß, daß die Intelligenz eines Mikrocomputers nur knapp über der eines Regenwurms angeordnet ist. Die hohe Leistungsfähigkeit der Mikrocomputer stammt allein von ihrer hohen Verarbeitungsgeschwindigkeit. Es kursieren über Computer allgemein verschiedene Sprüche, die den Sachverhalt auf humorvolle Weise wiedergeben. Ein bekannter Satz lautet zum Beispiel: "Computer sind doof, dafür aber wenigstens fleißig". Oft redet man auch davon, daß der "Computer ein Produkt menschlicher Faulheit ist".

Mikroprozessor. Personal Computer. Hobbycomputer. Mikrocomputer. Arbeitsspeicher. ROM. Peripherie. Minirechner. Einplatinencomputer. Entwicklungssystem. OEM - Computer - alles das sind Ausdrücke, mit denen man in der heutigen Zeit im Zusammenhang mit Computern zu tun hat - mit denen man zum

Beispiel beim Kauf eines Systems "vollgepumpt" wird von Verkäufern, die manchmal (oder meist?) selber nicht wissen, was sie erzählen. Um mehr von Computern zu verstehen, muß man zuerst die Struktur, das Grundprinzip von Computern, kennen.

Ein Computer besteht aus Speicher, Zentraleinheit und Ein/Ausgabe - Systemen, und kennt nur zwei Zustände. Das war schon immer so, und wird es bleiben - könnte man meinen! Aber nein, was die meisten nicht wissen: Es exisieren zwei Arten von Rechnern, der ANALOG- und der DIGITAL-Rechner. Der Analogrechner sei hier aber nur der Vollständigkeit halber erwähnt.

Uns wird hier nur der Digitalrechner beschäftigen. Die Sache mit den zwei Zuständen stimmt allerdings: Er ist entweder defekt oder in Ordnung. Ansonsten weiß zwar der Zuhörer, was mit den zwei Zuständen gemeint ist, aber richtig ist diese Aussage dennoch nicht. Ein "Von Neumann - Rechner", wie wir ihn in Form des ATARI zu Hause zu stehen haben, kennt zwar effektiv nur endlich viele Zustände, aber es sind doch immerhin noch eine ganze Menge! Doch dazu kommen wir später. Erst einmal wollen wir die Sache mit den zwei Zuständen ENDGÜLTIG klarstellen:

1). Das Zahlensystem, mit dem Digitalrechner arbeiten, kennt nur zwei Zustände.

Was heißt das? Unser Zahlensystem, nach dem wir dem Bäcker die Brötchen bezahlen, kennt zehn Zustände. Sie werden durch die zehn Ziffernsymbole 0,1,2,3,4,5,6,7,8,9 gekennzeichnet. Daraus lassen sich dann Kombinationen dieser Ziffern erstellen, die Zahlen genannt werden. Als Beispiel die Zahl 1234, eintausendzweihundertvierunddreißig (eigentlich ..dreißigundvier, aber das sind Eigenheiten der deutschen Sprache, genauso wie elf, zwölf usw.). Unter dieser Zahl kann man sich ab der 2. Schulklasse etwas vorstellen, man kann sogar damit arbeiten (rechnen) :

"1234 und 17 macht (4 und 7 macht 1, 1 im Sinn, 3 und 1 und 1 im Sinn macht 5, und 2 und 1 bleiben) 1251."

So rechnet man. Später, nach einiger Übung, geht das schneller, das Prinzip bleibt aber. Nur: WARUM rechnet man so? Oder anders gefragt: Rechnet man immer so? Die Antwort auf diese Frage lautet: N E I N .

Zum Beispiel: Wieviele Stunden liegen zwischen 18:00 und 03:00 Uhr?

Normalerweise würde man 3,00 - 18,00 rechnen und somit -15:00 herausbekommen. Es sind aber 9:00 Stunden - WARUM ?

Wer mitgelesen hat das Problem erkannt: Die Zeit läuft nicht nach dem Dezimalsystem mit 10 Ziffern, sondern nach dem Duodezimalsystem mit (eigentlich) 12 Ziffern. Denn eigentlich müßte man folgende Rechnung aufstellen: Der Tag endet bei 24:00. 24:00 + 3:00 = 27:00 , 27:00 - 18:00 = 09:00.

Wir sehen also, die Sache mit den Zahlensystemen ist reichlich konfus; speziell bei der Zeit, weil diese auf Grund der zehn benutzten Ziffern bei einem 12er-System die Vollendung des Zahlenchaos' bedeutet. Aber wir wollen nicht vergessen, daß die Engländer lange im 12er-System gerechnet haben, und gestorben sind sie daran nicht.

Weshalb sollte nun ein Computer mehr als zwei Ziffern kennen?

Sollten wir Mathematiker unter uns haben, so sei gesagt, daß sich genau berechnen läßt, mit welcher Anzahl von Ziffern die Effektivität des Zahlensystems am höchsten ist. Schlaue Leute haben sich die Mühe des Nachrechnens gemacht und kamen auf das Ergebnis, daß bei Verwendung von ca. 2,7 Ziffern die Effektivität am höchsten sei. Bloß- was soll die 0,7. Ziffer

darstellen? Normalerweise würde man den (mathematisch richtigen) Unsinn auf 3,0 erhöhen; man hätte dann die Ziffern 0,1 und 2 zur Verfügung, was sich heute technisch in Computern realisieren ließe (und auch teilweise in komplexen Zählersystemen angewandt wird), aber man darf nicht vergessen, daß es den Computer schon eine Weile gibt; damals war noch nicht einmal an den Transistor oder den konsequenten Einsatz von Röhren zu denken (mechanische Rechenwerke seien hier nicht betrachtet!).

Im Jahre 1939 bauten die Bell- Ingenieure ihren ersten Relais-Computer Model I.

Relais - wer kennt das nicht, das Gebilde aus einer Spule und einem Anker mit Schaltkontakten? Fließt Strom durch die Spule, bewegt sich der Anker und öffnet bzw. schließt die Kontakte.

H A L T !!!

Da war es schon:"... öffnet oder schließt die Kontakte" - das sind genau zwei klar definierte Zustände, die sich abfragen und generieren lassen! Also, wenn drei Zustände (das Optimum) schlecht zu erreichen sind, dann nehmen wir halt zwei. Dieser Gedanke hat uns prinzipiell den ATARI beschert!

Die zwei Zustände können auf verschiedene Arten auftreten: Zum Beispiel Relais angezogen und Relais abgefallen, Licht an und Licht aus, Taste gedrückt und Taste nicht gedrückt, Zusatzgerät vorhanden und Zusatzgerät nicht vorhanden, Spannung auf "HIGH-Pegel" und Spannung auf "LOW-Pegel", es fließt Strom, und es fließt kein Strom. Diese Gegenüberstellung läßt sich fast beliebig fortsetzen.

```
**************************
*                        *
*                        *
*  Von Bits und Bytes    *
*  ------------------    *
*                        *
**************************
```

Wir haben die Entstehungsgeschichte des Systems mit den zwei Ziffern kennengelernt; jetzt wollen wir damit rechnen.

Im Englischen heißt die Ziffer DIGIT. Hat man nur zwei Ziffern zur Verfügung, nennt sie der Engländer BINARY DIGIT, also binäre (zweiwertige) Ziffer, abgekürzt

B I T

Wir haben also jede Menge Bits - aber was damit anfangen? Was machen wir mit unseren zehn Ziffern? Wir 'hängen' sie nebeneinander und geben ihnen die ihnen zugehörigen Wertigkeiten:

1234 ist eine Zahl, hier sei sie aus dem Zehnersystem.

Das heißt, es sind

$$
\begin{aligned}
& 1 * 1000 \\
+\ & 2 * 100 \\
+\ & 3 * 10 \\
+\ & 4 * 1
\end{aligned}
$$

Nun wollen wir es noch einmal anders formulieren:

Jetzt sind es

$$
\begin{aligned}
& 1 * 10^{3} \\
+\ & 2 * 10^{2} \\
+\ & 3 * 10^{1} \\
+\ & 4 * 10^{0}
\end{aligned}
$$

Das ist logisch, denn 10 ** 3 (andere Formulierung für 10^3) = 1000, 10**2 = 100, 10**1 = 10 und schließlich: 10**0 = 1. Daß 10**0 = 1 ist, sollten wir uns besonders merken, denn es gilt allgemein:

Für JEDES "a" aus der Menge der reellen Zahlen gilt:

a ** 0 = 1

Diesen Satz benötigen wir später wieder, also MERKEN !!!

So, und nun setzen wir einfach mehrere BInary DigiTs aneinander:

1101 0111

(der Lesbarkeit halber in zwei Vierergruppen aufgeteilt). Das ergibt dann, ebenfalls von links nach rechts aufgeschlüsselt:

```
      1 * 2**7 = 1 * 128 = 128
    + 1 * 2**6 = 1 *  64 =  64
    + 0 * 2**5 = 0 *  32 =   0
    + 1 * 2**4 = 1 *  16 =  16
    + 0 * 2**3 = 0 *   8 =   0
    + 1 * 2**2 = 1 *   4 =   4
    + 1 * 2**1 = 1 *   2 =   2
    + 1 * 2**0 = 1 *   1 =   1
      ---------------------------
                             215
```

Jetzt haben wir unsere Zahlen, deren Ziffern (Bits) nur zwei Zustände kennen. Die Aufteilung der Zahlen in Gruppen zu mehreren Ziffern (hier Vierergruppen) ergibt sich auch bei unserem Dezimalsystem, denn wie schreibt der Kaufmann die Million?

1.000.000,--

Hier haben wir die Aufteilung der Zahl in Dreiergruppen, hervorgerufen durch den Dezimalpunkt (im Deutschen - im Englischen sind Dezimalpunkt und - Komma in der Bedeutung genau vertauscht, also nicht durch englische Literatur irritieren lassen!). Die Aufteilung der Binärzahlen kann auch in Dreiergruppen oder sonstwie erfolgen, durchgesetzt bei den informatorischen Wissenschaften haben sich jedoch nur Dreier- und Vierergruppen.

Die Zahl 1101 0111 liest sich natürlich etwas langatmig: "einseinsnulleinsnulleinseinseins" - das wäre sehr umständlich. Was ist also zu tun? Man überläßt dem Computer die Fieselarbeit mit den Nullen und Einsen, als gebildeter Informatiker jedoch unterhält man sich 'Hex', d.h., Hexadezimal (Genaugenommen müßte man von "Sedezimal" reden, doch dieser Begriff ist nur selten anzutreffen.). Was bedeutet das?

Wir hatten unsere Zahl in zwei Vier-Bit-Gruppen eingeteilt, und wie man leicht ausprobieren kann, hat man bei vier Stellen und jeweils zwei Möglichkeiten pro Stelle insgesamt SECHZEHN Kombinationsmöglichkeiten.

Diese wären

0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111,
1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111 .

Jetzt wollen wir diesen sechzehn Kombinationsmöglichkeiten einmal Namen geben :

0000	soll heißen :	0	und bedeutet dezimal	0
0001		1		1
0010		2		2
0011		3		3
0100		4		4

0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Auf unsere Zahl bezogen können wir nun sagen:

1101 0111 heißt in Hexadezimalform : D 7 .

Wir haben es hier also mit der Zahl D7 zu tun. Aber wie erhält man nun aus D7 eine Dezimalzahl?

Ganz einfach, wir wandeln die Zahl D7 wieder in 1101 0111 um und berechnen nach der oben gezeigten Methode die Dezimalzahl 215. Gibt es auch noch einen anderen, vielleicht einfacheren Weg, diese, und vor allem größere Hex-Zahlen umzuwandeln? Sicherlich gibt es den, und der aufmerksame Leser kennt oder ahnt ihn zumindest schon: Wir haben es hier ganz offensichtlich mit zwei Ziffern aus dem Hexadezimalsystem zu tun, können also sagen:

```
D7 =   D * 16**1  = 13 * 16**1 = 13 * 16 = 208
     + 7 * 16**0  =  7 * 16**0 =  7 *  1 =   7
     -----------------------------------------
                                           215
```

Wir kennen nun die Umrechnungsart von binär oder hexadezimal nach dezimal; und wie verhält es sich umgekehrt? Hier gibt es mehrere Methoden. Als erstes könnte man die Dezimalzahl in eine binäre und die dann, durch einfache Aufteilung in

Vierergruppen von rechts nach links, in eine Hexzahl wandeln. Da dazu jedoch eine ganze Reihe von Divisionen notwendig ist, empfiehlt sich diese Art nur dann, wenn man nur seinen Kopf zum Rechnen hat.

Wir nehmen uns wieder die dezimale 215:

```
215 / 2 = 107      Rest 1
107 / 2 =  53      Rest 1
 53 / 2 =  26      Rest 1
 26 / 2 =  13      Rest 0
 13 / 2 =   6      Rest 1
  6 / 2 =   3      Rest 0
  3 / 2 =   1      Rest 1
  1 / 2 =   0      Rest 1
```

Unsere Binärzahl besteht aus den Restziffern, von UNTEN NACH OBEN gelesen!

Aber wie gesagt, die Methode läßt sich auch am Montagmorgen im Kopf ausführen, die nun folgende geht wesentlich schneller, erfordert aber Rechenarbeit, speziell bei großen Zahlen:

```
215,00 / 16 = 13,44 (Dez)  = D (Hex)
  0,44 * 16 =  7,04 "      = 7 "
```

Also: durch die größte Potenz von 16, die kleiner ist als die zu wandelnde Zahl teilen, und dann den jeweiligen Rest so oft mit 16 muliplizieren, wie die Potenz betrug, durch die wir zu Beginn geteilt hatten.

Aber das sollten wir besser noch einmal an einem etwas größeren Beispiel ausprobieren:

Die Zahl 60413 (Dez) wollen wir wandeln. Die nächstkleinere 16er-Potenz ist 16**3 = 4096 (denn 16**4 = 65536, also viel größer!).

```
0). 60413 / 4096 = 14,7493 ==: 14 (Dez)  = E (Hex)
1). 0,7493 *  16 = 11,9888 ==: 11   "    = B   "
2). 0,9888 *  16 = 15,8208 ==: 15   "    = F   "
3). 0,8208 *  16 = 13,1328 ==: 13   "    = D   "
```

Hier hören wir auf, denn unsere Sechzehnerpotenz betrug 3 (zur Erinnerung, wir hatten als Erstes durch 16**3 = 4096 geteilt!), und wir haben danach 3 Mal mit 16 multipliziert.

Unsere Hexadezimalzahl steht nun deutlich da: EBFD . Zur Überprüfung machen wir die Probe :

Es gilt, die Hexzahl EBFD in eine Dezimalzahl zu wandeln:

```
EBFD =   E * 16**3 = 14 * 16**3 = 14 * 4096 =  57344
       + B * 16**2 = 11 * 16**2 = 11 *  256 =   2816
       + F * 16**1 = 15 * 16**1 = 15 *   16 =    240
       + D * 16**0 = 13 * 16**0 = 16 *    1 =     13
       ------------------------------------------------
                                               60413
```

Die Probe zeigt also das gleiche Ergebnis.

Wenn man das mehrfach übt, und sich einige Übungsaufgaben stellt, beherrscht man diese Umrechnungsarten recht schnell. Wer dazu allerdings keine Lust verspürt, kann sich ein geeignetes Programm schreiben...

Denn diese Umrechnungen von Dez nach Hex und umgekehrt werden uns auf dem Atari noch eine ganze Weile nachlaufen, denn das Atari-BASIC versteht nur Dezimalzahlen!!! Wer allerdings wissen will warum, der möge die Götter fragen, denn eines wissen wir inzwischen genau: Der Computer mag wählen zwischen binärer oder hexadezimaler Darstellung, Dezimal rechnet er auf gar keinen Fall!

Wir bezeichnen in unserem Buch übrigens die Hexadezimalzahlen mit einem "$". Die "d7" hieße also "$d7".

Die Bits haben wir nun kennengelernt, jetzt beschäftigen wir uns mit den

B y t e s .

Dieser Ausdruck, ist schnell erklärt. Wir haben vorhin 8 Bits, aufgeteilt in zwei Vierergruppen, genannt NIBBLEs (sprich: Nibbel), betrachtet. Warum denn gerade acht, warum nicht sieben oder neun? Nun, da wir uns geeinigt hatten, Vierergruppen zu verwenden, ist es schon nicht unpraktisch, diese Vierergruppen jeweils "voll" zu machen. Außerdem "holt" sich der ATARI immer 8 Bits auf einmal zum Arbeiten; warum sollten wir nun auch wieder anders "denken" als unser großer Bruder? In der Computerbranche hat sich für diese 8 gleichzeitigen (paralelen) Bits der Name BYTE eingebürgert. Er bedeutet BINARY TERZ also binäre Dreiergruppe. Diese Bezeichnung wurde von einer großen Firma in den USA geprägt. Sie erklärte ihren Programmierern nicht das Hexadezimal- sondern das Oktalsystem, bei dem immer drei Bits zusammengefaßt werden.

Wir merken uns : Ursprünglich bedeutete "BYTE" im Oktalsystem die Zusammenfassung von drei Bits.

Jetzt hingegen gilt:

1 Byte besteht aus 8 Bits (Hexadezimalsystem)

```
*****************************************
*                                       *
*                                       *
*  Die HARDWARE eines Mikrocomputers    *
*  ---------------------------------    *
*                                       *
*****************************************
```

Im vorigen Kapitel hatten wir gesagt, woraus ein Computer besteht:

Speicher
Zentraleinheit
Ein-/Ausgabeleitungen

Diese drei Komponenten findet man aber nicht in nur drei Bausteinen, es befinden sich erheblich mehr ICs (Integrated Circuit, integrierter Schaltkreis) im Atari. Der schaltungstechnische Aufwand ist nur damit zu begründen, daß es noch keinen Baustein gibt, der den ganzen, im ATARI zur Verfügung stehenden Speicherplatz z.B. ausfüllt; bei der Zentraleinheit und den I/O (INPUT/OUTPUT) - Leitungen verhält es sich genauso. Man muß nämlich bedenken, daß alle im ATARI eingebauten Bausteine auch allgemein anwendbar sind, und somit zur Ein- und Ausgabe mehrere "Interfacebausteine" gehören.

D E R   S P E I C H E R

Wir wollen hier prinzipiell zwei Arten des Speichers unterscheiden :

1) den Schreib / Lesespeicher

und

2) den Nur - Lesepeicher. (ROM bzw. Read Only Memory)

Den Schreib / Lesespeicher nennt man auch Neudeutsch "RAM" (sprich: Rämm), was "Random Access Memory" bedeutet, oder "Speicher mit wahlfreiem Zugriff". Mit Zugriff wird hier nicht der Zugriff auf eine bestimmte Adresse gemeint; der ist für RAM wie ROM gleich. Es soll hier vielmehr eine Unterscheidung getroffen werden, ob es sich um einen Schreib- oder Lesezugriff handelt. Diese Speicherart kann man sich vorstellen als eine Kommode mit mehreren Schubladen. Nehmen wir an, unsere Kommode hätte sechs Schubladen mit zwei Reihen zu je drei Stück:

Dann können wir die sechs Schubladen auf zwei Arten bezeichnen:

```
1)     ----------------------------------------
       I           I           I           I
       I     0     I     1     I     2     I
       I           I           I           I
       ----------------------------------------
       I           I           I           I
       I     3     I     4     I     5     I
       I           I           I           I
       ----------------------------------------
```

Damit hätten wir die Schubladen 0 ... 5, die wir nun eindeutig unterscheiden können. Es gibt auch noch eine andere Methode:

```
2)     ----------------------------------------
       I           I           I           I
     0 I     0     I     1     I     2     I
       I           I           I           I
       ----------------------------------------
       I           I           I           I
     1 I     0     I     1     I     2     I
       I           I           I           I
       ----------------------------------------
```

Hiermit erhalten wir die Namen (0,0), (0,1), (0,2),
(1,0), (1,1), (1,2) .

Auch hiermit lassen sich alle Schubladen eindeutig bezeichnen.

Wir wollen uns für die Zukunft diese beiden Arten der Numerierung merken: Die erste nennt man LINEARE ANORDNUNG, die zweite versehen wir mit dem Begriff MATRIXFÖRMIGE ANORDNUNG.

Hardwaremäßig gesehen, arbeiten Speicher ab einer bestimmten Größe matrixförmig; aber softwaremäßig werden die Speicherstellen linear angesprochen (verwirrender Weise auch wenn man Matritzen programmiert - aber damit kämen wir vom Thema ab).

Wir haben also in unserem Falle sechs Schubladen, die wir unterscheiden können. Nun wollen wir sie aber auch benutzen. Wir öffnen Schublade Null und sehen nach, was darin ist :

NICHTS

Warum ? Wir haben ja noch nichts hineingetan. Aber ist Nichts denn wirklich NICHTS ? Die Frage ist ein bißchen philosophisch, deshalb wollen wir einmal ein wenig vom Thema abschweifen, um drei äußerst wesentliche informationstechnische Begriffe zu klären, mit deren Hilfe sich die oben gestellte Frage (für unsere Zwecke!!!) leicht lösen läßt.

Der Informatiker unterscheidet drei Merkmale einer Information:

1) Die syntaktischen,
2) die semantischen und
3) die pragmatischen Aspekte einer gegebenen Information.

Was bedeuten diese Fremdworte?

Unter "SYNTAX" versteht man den prinzipiellen Aufbau einer Information.

Nehmen wir ein beliebiges Wort: zum Beispiel " M A N N ". Würden wir zwar MANN meinen, aber MAN schreiben, verstünde uns keiner. Also halten wir uns an die Regeln (Syntaxregeln!) der deutschen Sprache und schreiben "MANN".

Als anderes Beispiel sei hier die Verkehrsampel gewählt: Die Syntax der sog. "Wechsellichtanlage" lautet:

rot
rot + gelb
grün
gelb

Würden Rot und Grün zusammen erscheinen, wüßte niemand etwas mit dieser Information anzufangen - sie wäre SYNTAKTISCH FALSCH.

Unter SEMANTIK versteht man den Informationsgehalt, den sich der Entsender der Information gedacht hat oder, anders formuliert, das, was normalerweise bei Eintreffen der Information daraufhin passieren sollte.

Der semantische Aspekt der roten Ampel wäre :

"HALT".

Die dritte Betrachtung einer Information, die PRAGMATIK bezeichnet das, was den Nutzer der Information zu deren Nutzung treibt, oder wie er eine Information für sich speziell verwendet. Sieht man z.B. eine gelbe Ampel, denkt man sich "Stoff, dann schaff' ich sie noch", oder "hat sowieso keinen Zweck mehr, Gas weg".

Mit diesen Definitionen läßt sich die an sich philosophisch nicht zu lösende Frage informationstechnisch relativ leicht beantworten. "Wir sehen in der soeben geöffneten Schublade nichts"; soweit waren wir schon.

NICHTS, WAS WIR SINNVOLL VERWERTEN KÖNNEN

Nach unseren drei Punkten aufgeschlüsselt:

1) Die Syntax der Information stimmt; irgendetwas ist sicherlich in der Schublade, und wenn es ein Vakuum (luftleerer Raum) ist.

2) Die Semantik ist auch klar, wir haben noch nichts hineingepackt, also können wir auch nichts herausholen.

3) Pragmatisch betrachtet ist die Schublade für uns leer (obwohl etwas, nämlich NICHTS darin ist!), wir können also irgendetwas hineintun. Tun wir es, und nehmen wir oder ein anderer nicht wieder etwas heraus, werden wir es immer wieder nach Öffnen der Schublade finden.

Jetzt formulieren wir den eben beschriebenen Vorgang technisch:

Wir nennen die Nummern, mit denen wir die Schubladen bezeichnet haben,

A D R E S S E N

In jede Schublade passen GENAU acht Bits hinein (also weder mehr noch weniger, eben: genau acht!).

Jetzt öffnen wir eine Schublade, nehmen wir als Beispiel die Nummer 3. Diesen Vorgang des Öffnens der Schubladen wollen wir in Zukunft

A D R E S S I E R E N

nennen,

das Hineinsehen soll L E S E N ,
das Hineinlegen soll S C H R E I B E N ,
die Schublade selber soll S P E I C H E R P L A T Z
heissen.

Das sind zwar eine ganze Menge Namen auf einmal, aber mit der Zeit werden wir lernen, richtig damit umzugehen.

Also, wir ADRESSIEREN SPEICHERPLATZ 3, und LESEN ihn aus. Was steht darin? Nichts ist die Antwort, denn wir haben ja noch nichts hineingeschrieben.

Wie wir im vorigen Kapitel erfahren haben, können nur Nullen und Einsen in dem Speicherplatz stehen, denn wir haben ja nur zwei Ziffern, und außerdem hatten wir eben gesagt, daß nichts nicht unbedingt nichts sein muß. Es könnte also sein, daß z.B. die Zahl 0110 1100 (6c Hex, $6c) darin enthalten ist - wieso auch nicht?!?

Technisch läßt sich das so erklären, daß durch interne Unebenheiten der in den Bauteilen enthaltenen Leiterbahnen beliebige Bitkombinationen nach dem Einschalten enthalten sind. Und diese Bitkombinationen können also alle 256 Möglichkeiten der Kombination von acht Nullen und Einsen sein, nämlich 0000 0000 bis 1111 1111 (es gibt zwei Möglichkeiten, entweder man glaubt uns, daß es 256 Kombinationen sind, oder man berechnet die Anzahl - letzteres wäre eine gute Übung zum vorherigen Kapitel!).

Nehmen wir also an, wir hätten einen Computer vor uns, bei dem nach dem Einschalten in Speicherplatz 3 die Hexzahl $6C steht.

Diese können wir natürlich jederzeit wieder lesen, denn sie steht ja drin. Nur, der pragmatische Teil dieser Information ist für uns als Benutzer unwesentlich größer als Null, denn es ist eine reine ZUFALLSINFORMATION !!!

Also wollen wir etwas Vernünftiges hineinschreiben. Dazu wählen wir unsere Hexzahl aus dem vorigen Kapitel, die "$d7". Wir haben unseren Speicherplatz Nummer 3 immer noch adressiert, und SCHREIBEN nun die Zahl $d7 hinein.

Was geschieht?

Die Zufallszahl $6c räumt den Speicherplatz 3 und es wird die $d7 GESPEICHERT. Wenn wir nun später wieder auf Speicherplatz Nummer 3 zugreifen, werden wir immer wieder die Zahl $d7 lesen, bis wir eine neue Zahl in die Speicherstelle schreiben oder den Rechner ausschalten.

Im ersten Fall wird selbstverständlich die $d7 von der neuen Zahl ÜBERSCHRIEBEN im zweiten Fall erhalten wir wieder unsere Zufallsinformation.

Gibt es denn keine Möglichkeit, eine einmal eingespeicherte Information auch nach dem Abschalten zu erhalten oder vor dem Überschreiben zu sichern?

Nur durch schaltungstechnische Eingriffe in den Computer; das RAM ist eben ein Schreib / Lesespeicher. Wollen wir einen Speicher haben, der zwar (für uns verwertbare) Informationen hergibt, sie aber nicht überschreiben kann, müssen wir Nur - Lesespeicher verwenden. Um die Informationen im RAM beim ausgeschalteten Rechner zu schützen, müßte man die RAMs auch nach dem Abschalten aller anderen Teile des Rechners weiter mit Strom versorgen. Zu diesem Zweck gibt es auch RAMs die nur sehr wenig Strom verbrauchen. Im Atari wurden jedoch "normale" RAMs verwendet.

Die Nur-Lesespeicher werden ROM (Read Only Memory) genannt. Es gibt viele unterschiedlich Arten von ROMs, die sich in der Bezeichnung immer nur durch Buchstaben VOR dem "ROM" unterscheiden, z.B. EPROM, EEROM, EAROM, PROM, IPROM. In der Bezeichnung der Hersteller "heißt" ein bestimmtes EPROM aber zum Beispiel "2716".

Alle haben aber eines gemeinsam: Der Computer kann die in ihnen enthaltene Information nicht OHNE HILFSMITTEL ändern, bei einigen Bausteinen wird die Information sogar gleich bei der Entwicklung des Types hineingegeben, sodaß sie von uns niemals mehr änderbar ist (außer natürlich durch Zerstörung des Bausteines).

D I E Z E N T R A L E I N H E I T

Die Zentraleinheit übernimmt die gesamte Verwaltung des Computers. Sie adressiert den zur Verfügung stehenden Speicher, behandelt Daten in dem Speicher und verwaltet bzw. dirigiert die I/O - Leitungen. Sie ist also das Kernstück eines jeden Computers. Ist sie defekt, läuft der ganze Rechner nicht mehr.

Allgemein besteht sie aus einigen internen Speicherplätzen, sogenannten Registern, die im Gegensatz zum übrigen Speicher den Vorteil haben, daß sie wesentlich schneller von der Zentraleinheit angesprochen werden können.

Außerdem sind in der CPU (Central Processing Unit, Zentrale Prozesseinheit) noch eine komplexe Logik zum Rechnen, Treiber zum Ansteuern der Speicher und I/O - Leitungen sowie einige interne, nicht direkt adressierbare Zwischenspeicher enthalten.

Nach außen hin liefert die CPU Adressensignale zum Ansteuern der weiteren Baugruppen, Datenleitungen zum Transport (Lesen oder Schreiben) von Daten sowie einige Kontrollsignale, die den Status der CPU angeben, so z.B. eine Schreib/Leseleitung, die angibt, ob Daten gesendet oder gelesen werden sollen.

In unserem ATARI haben wir es mit der 6502 - CPU (lies: fünfundsechzignullzwo zehpehuh), bzw. der 6502C - CPU zu tun. Sie besitzt u.a. 8 Datenleitungen, mit denen sie die jeweils 8 Bit vom oder zum Speicher transportieren kann.

Weiterhin liefert sie 16 Adressensignale, mit denen man Speicher oder I/O - Leitungen ansprechen kann. 16 Leitungen also, wieviele Speicherplätze sind das denn? Wer im vorigen Kapitel aufgepaßt hat, hat die Antwort schnell parat: es sind genau 2**16 Speicherstellen oder I/O - Adressen.

Hier sei noch folgende Konvention der Techniker erwähnt :

Im Gegensatz zum "normalen" Kilo, das einem Faktor von 1000 entspricht (man denke z.B. an das Kilogramm: 1kg = 1000g), spricht der Informatiker von dem Kilo als 1024.

Also : 1 KBit = 1024 Bit

Die "krumme" Zahl 1024 läßt sich leicht begründen: 1024 = 2**10. Wenn man also von einem Kilobyte spricht, meint man 1024 Byte. 16 KByte sind 16384 Bytes und 64 KByte sind 65536 Bytes.

DAS IST KEIN DRUCKFEHLER!

Genau bei 64 Kilo stimmt die Daumenregel Anzahl mal tausend nicht mehr! Also nicht verwirren lassen, z.T. zum Verkauf angebotene Computer mit 65 KByte gibt es nicht! Leider wird selbst in Elektronikzeitschriften gelegentlich von 65 KByte gesprochen.

Wer bis hierhin alles verstanden hat, ist auch sicherlich in der Lage, zu erklären, weshalb es keinen Computer geben KANN der GENAU 65 KByte adressiert.

Die Begründung ist einfach: 16 Adressleitungen adressieren 64 KByte RAM, 17 Adressleitungen dagegen schon 128 KByte. Wieviele Adressleitungen bräuchte man also für 65 KByte Adressraum ?

Die 64 KByte Adressraum der CPU werden nun aufgeteilt in RAM, ROM und

I N P U T / O U T P U T - L E I T U N G E N

Unter diesem Begriff versteht man nicht bloß eine Anordnung von Leitungen, über die Daten transportiert werden, sondern insbesondere die zu deren Ansteuerung notwendige Elektronik, die aus den Computer-Bits elektromechanische, elektronische, optische oder akustische Signale erzeugt, und somit die vom Computer gegebenen Informationen hörbar, sichtbar oder für ein beliebiges Medium speicherbar macht. Diese Wandler werden gemeinhin als "INTERFACES" oder "CONTROLLER" bezeichnet.

Unter einem INTERFACE versteht man eine Baugruppe, die die von einem beliebigen Geber gesendeten Signale so umwandelt und weitersendet, daß ein ebenfalls daran angeschlossener Empfänger sie erkennen, kontrollieren und verarbeiten kann.

Ein CONTROLLER "bewacht" das ihm schutzbefohlene Peripheriegerät (Peripherie = griech. Umfeld, Umfeld, also das, was "drumrum" ist wie z.B. Drucker, Floppy o.ä.) und versorgt es mit Befehlen und Daten beziehungsweise holt Daten von ihm ab.

Selbstverständlich gehört zu der jeweiligen Elektronik auch immer eine Software, der TREIBER, der das sogenannte PROTOKOLL, das den Ablauf der Datenübertragung regelt, festlegt. Er reagiert zum Beispiel auf eine Datenanfrage, einen Fehler in einer Übertragung oder ähnliches mehr. Einige Protokolle sind standardisiert, oder ihre Standardisierung ist in Vorbereitung. Zuständig hierfür sind Institutionen wie die ISO (International Standardisation Organisation) oder CCITT, das französische Gegenstück.

```
*****************************
*                           *
*                           *
* L O W   und   H I G H     *
* ----------------------    *
*                           *
*****************************
```

Uns interessiert jetzt die Frage, wie sich der Computer mit dem Speicher und den peripheren Geräten unterhalten kann, denn selbst bei Aufschrauben des Computers sind nirgends Nullen oder Einsen zu finden (Schlaumeier aufgemerkt, wer meint, die 6502 - CPU enthalte eine Null, hat irgendetwas nicht richtig verstanden...). Auch die am Anfang beschriebenen Relais, die offen oder geschlossen gewesen sein konnten, sind nicht im ATARI zu finden; wo steckt dann die Information?

Sie wird in Form von elektrischen Spannungen übertragen, die im Interesse der einzelnen Computerfirmen standardisiert sind. Es gibt dafür allerdings mehrere unterschiedliche Arten elektronischer Baugruppenstandards, aber uns interessiert hier nur einer:

DER TTL - STANDARD

Alle Bausteine auf TTL - Basis (Transistor - Transistor - Logik) arbeiten mit einer Versorgungsspannung von +5 Volt, sofern sie nicht für den Interfacebetrieb an höheren Spannungen entwickelt wurden. Die Versorgungsspannung ist die Spannung, die einen Baustein zum "Leben" erweckt, ohne die sich "nichts tut".

Ein anderes Herstellungsverfahren als das der TTL - Logik, das uns die MOS (Metal Oxyd Semiconductor, Metalloxyd - Halbleiter) - Bausteine "schenkte", lieferte uns auch die 6502 - CPU sowie alle höher integrierten Bausteine, wie z.B. die Speicher.

Unter höherer Integration verstehen wir, daß auf gleichem Platz (Quadratmillimeter) mehr Bauteile untergebracht sind - und solche Bausteine wie Speicher oder CPUs "verbraten" immense Mengen von Bauteilen. Wollte man z.B. die CPU mit einzelnen Transistoren aufbauen, bräuchte man schon einen kompletten mittleren Wohnzimmerschrank inclusive Kühlanlage! Der ist aber, wie gesagt, nicht (mehr) notwendig.

Betrieben werden unsere Bausteine also mit einer Versorgungsspannung von +5 Volt gegen Massepotential (entspricht meist dem Erdpotential). Jetzt können wir zwei unterscheidbare Zustände definieren:

1) der LOW (niedrigere) - Zustand soll der negativeren und
2) der HIGH (höhere) - Zustand soll der positiveren Spannung entsprechen.

Warum so kompliziert, warum nicht einfach LOW=Masse und HIGH=+5V? Diese komplizierte Ausdrucksweise ist notwendig, da in (nicht TTL-) Baugruppen Versorgungsspannungen von z.B. -5V existieren können. Hier wäre dann eine neue Definition notwendig, die man mit der komplizierten Beschreibung oben aber ebenfalls abgedeckt hat. Bei TTL-Bausteinen sind die Potentiale der Pegel folgendermaßen definiert :

HIGH - Potential liegt zwischen
+2,4 Volt und +5,0 Volt

und

LOW - Potential liegt zwischen
0 Volt und +0,8Volt

Was liegt aber zwischen +0,8 und 2,4 Volt Eingangsspannung?

Bitte nicht lachen: Das weiß der Baustein, der diese Eingangsspannung empfängt, selbst nicht!

Im Allgemeinen fangen ICs bei unklaren Eingangspegeln an zu schwingen, das heißt, sie erkennen in schnellem Wechsel am Eingang abwechselnd LOW und HIGH.

Im übrigen gilt diese Definition oben von Low und High nicht immer; wir sprechen hier von POSITIVER LOGIK. Bei der negativen Logik ist alles genau umgekehrt, aber darauf wollen wir jetzt nicht weiter eingehen.

Jetzt wissen wir, wie der Computer seine Daten haben möchte: in Form von Spannungen. Spannungen kann man nicht sehen, nur ihre Auswirkungen (z.B. Blitze!) . Man kann aber auch Geräte benutzen, die uns das Vorhandensein oder Fehlen von Spannungen anzeigen. Da wären als einfachste Geräte die Logikprüfstifte zu nennen.

Sie bestehen aus einem kugelschreibergroßen Gehäuse, in dem sich zwei oder mehr Lampen befinden; je nachdem, welche der Lampen leuchtet, liegt die eine oder die andere Spannung an. Dies ist das billigste Meßinstrument in der Digitaltechnik, nur leider arbeitet unser ATARI so schnell, daß wir in den meisten Fällen beide Lämpchen leuchten sehen würden, da sich die Pegel an den ICs 50000 Mal so schnell ändern wie wir sehen können! Also ist dieses Meßinstrument für uns meist zwecklos.

Als nächstes gibt es das sogenannte Vielfachmeßinstrument. Es kann mit einem Zeiger oder einer Digitalanzeige versehen sein - man benötigt es zwar manchmal zum Reparieren von Computern, aber prinzipiell ist auch dieses Instrument für die Anzeige "nicht-statischer" Spannungen und Ströme in einem Mikrocomputer zu langsam.

Das Gerät, mit dem man sich an den Computer "heranwagen" darf, ist das Oszilloskop (manchmal auch "Oszi" oder "Oskar" genannt) mit einer Minimalfrequenz von 40MHz (Megahertz).

Eine derartig hohe Frequenz (der Aatri arbeitet mit 1,77 MHz) wird benötigt, da wir digitale Signale messen wollen. Um diese richtig zu sehen, brauchen wir eine wesentlich höhere Frequenz als die des zu messenden Signals, denn Rechteckschwingungen bestehen aus beliebig vielen ungeradzahligen Harmonischen, und je mehr man davon sieht, umso besser werden die Signale auf dem Oszischirm gezeigt.

Diese kleine Ausschweifung in das (riesige) Feld der Meßtechnik sei als WARNUNG hier eingebaut.

> "Niemals mit ungeeigneten Meßinstrumenten versuchen, etwas zu reparieren! Das kann nur "schief" gehen."

Es gibt zwar für jeden Techniker einmal die Situation, daß er ein Gerät reparieren muß, ohne die geeigneten Meßgeräte dabei zu haben, aber :

> "Ein guter Techniker weiß, wann er aufhören muß zu improvisieren !"

Diesen Spruch sollte sich jeder über's Bett nageln, der an seinem ATARI herumschraubt, ohne GENAU zu wissen, was er macht. Damit wollen wir die kurze Einführung das "Prinzip eines Mikrocomputers" beenden.

D A S K O N Z E P T D E S A T A R I

Die Atari-Mikrocomputer sind vom Konzept her eindeutig als Hobbygeräte entworfen. Ihre Hard- und Softwareeigenschaften bieten verschiedene Anwendungen besonders an. So eignen sich die Atari-Geräte durch ihre Ton- und Bildeigenschaften besonders zum Videospielcomputer. Natürlich sind sie ebenso zum Erlernen von BASIC geeignet. Aber schon das BASIC ist mehr auf spielerische Anwendungen, als auf kommerzielle, ausgelegt. Genauso ist es möglich, mit Zusätzen auch andere Programmiersprachen, wie zum Beispiel Assembler oder auch PILOT, PASCAL, C oder ACTION!, kennenzulernen. Auch für eine Sprache wie Forth eignet sich der Atari hervorragend. Eine weitere Anwendung wäre die Verwendung des Atari als Lerncomputer für den Schulstoff (zum Beispiel für Vokabeln). Sicher kann man den Atari auch im Haushalt verwenden. So ließe sich beispielsweise eine Budgetüberwachung realisieren oder eine Schallplattenkartei aufbauen. Auch kürzere Briefe ließen sich durch den Atari leicht und effizient erstellen.

Jedoch erreicht der Atari hier seine Grenze. Größere Datenmengen lassen sich mit ihm nur sehr schwer bewältigen, da seine Diskettenlaufwerke nicht groß und schnell genug sind. Auch ist es mühsam, ihn als Textsystem zu gebrauchen, da sein Bildschirm nicht über die, für die meisten Zwecke benötigten, 80 Zeichen pro Zeile verfügt. Für kommerzielle Anwendungen sind die Atari-Geräte also nur sehr bedingt geeignet. Auch für "diskettenorientierte" Programmiersprachen, wie PASCAL, FORTRAN oder insbesondere COBOL, ist der Atari in der praktischen Arbeit nicht unbedingt geeignet.

Um den Atari in seinen Einzelheiten zu verstehen, sollte man sich sein Konzept, seine prinzipielle Struktur vor Augen führen. Erst wenn man diese "durchschaut" hat, ist es sinnvoll auf die Einzelheiten der hochintegrierten Peripheriebausteine (POKEY, ANTIC, GTIA und PIA) und des Betriebssystems einzugehen.

Bei den Atari-Geräten handelt es sich um Mikrocomputer, die mit dem weit verbreiteten Mikroprozessor "6502" ausgerüstet sind.

Sicherlich kann man sich angesichts der auf dem Markt befindlichen Prozessoren fragen, ob dieser Prozessor überhaupt noch zeitgemäß ist. Gerade die 16-Bit bzw. 32-Bit-Prozessoren sind erheblich leistungsfähiger. Auch gibt es in diesen Prozessorfamilien Bausteine, die über einen nur 8 Bit breiten Datenbus verfügen, die also kaum eine komplexere Hardware benötigen, als der 6502. Zur Zeit, als die ersten Atari-Mikrocomputer entwickelt wurden, gab es diese Prozessoren allerdings noch nicht in den benötigten Stückzahlen auf dem Markt. Auch sind sie noch heute wesentlich teurer als der 6502.

Andere zu dieser Zeit verfügbare 8-Bit-Prozessoren, wie zum Beispiel der Z80, der 8080/8085, der 6800/6802 oder auch der 6809, haben entweder keinen entscheidenden Leistungsvorsprung gegenüber dem 6502 oder sind selbst relativ teuer (oder waren zum Zeitpunkt der Entwicklung der ersten Atari-Geräte relativ teuer). Man sollte aber ehrlich anmerken, daß bei der Auswahl auch ganz andere Gesichtspunkte eine Rolle spielen. So kommt es bei derartigen "Großprojekten" (in der Stückzahl) auch auf das Vertrauen, und besonders die Verbindungen zum Lieferanten an. Auch müssen die personellen Möglichkeiten für eine Entwicklung vorhanden sein oder entsprechende Entwickler angeworben werden. Sicher spielen besondere Vorlieben der maßgeblichen Leute für einen bestimmten Prozessor auch noch eine Rolle.

Bei der Entwicklung der 600XL- und 800XL-Geräte wählte Atari wiederum einen 6502-Prozessor. Es wurde eine weiterentwickelte Version der bekannten CPU mit der Bezeichnung 6502C, die aber vollständig softwarekompatibel ist, verwendet. Dies geschah wohl vor allem mit Blick auf eine größtmögliche Kompatibilität zu den alten Modellen. So konnte man viele Programme und Programmteile direkt übernehmen.

Der 6502 hat außerdem den Vorteil, daß sein Systemtakt aus zwei gleichlangen Phasen besteht. Der Prozessor greift je-

weils nur während der zweiten Phase auf die Peripherie bzw. auf den Speicher zu. Die Zugriffe erfolgen also immer an den gleichen "Stellen" des Systemtaktes. Dieses Konzept vereinfacht die Hardware anderer Bausteine im System, die ohne Umweg über die CPU direkt auf den Speicher zugreifen sollen und dazu die CPU stoppen müssen. In diesem Detail unterscheidet sich der 6502 wesentlich von anderen Mikroprozessoren wie zum Beispiel vom Z80, 8080 oder auch 8085.

Der 6502-Prozessor wird im Atari mit 1,77 MHz betrieben. Diese Frequenz liegt erheblich über der, bei 6502-Rechnern dieser Preisklasse üblichen Arbeitsfrequenz.

Die normale Eingabe erfolgt über eine Tastatur. Die Belegung entspricht im wesentlichen der in Amerika üblichen Form (QWERTY). Einige Sonderzeichen sind beim Atari jedoch an anderer Stelle zu finden als bei anderen Computern. Die Umlaute oder andere, für andere Sprachen nötige Sonderzeichen oder Sonderformen von Buchstaben, sind beim Atari nicht ohne weiteres über die Tastatur verfügbar. Dies ist bei Computern in dieser Preisklasse aber leider üblich zu nennen. Der Atari 400 war mit einer Folientastatur ausgestattet. Das größere Modell, der Atari 800, besaß dagegen eine Tastatur mit "normalen" Tasten (ähnlich Schreibmaschinentasten). Die neuen Atari-Geräte, also der Atari 600XL und der Atari 800XL, verfügen grundsätzlich über "normale" Tasten. Die Tastaturbelegungen der Geräte unterscheiden sich nur in sehr kleinen Details, es gibt also nicht, wie bei anderen Herstellern, für jeden neuen Computer auch eine neue Tastaturbelegung. Zusätzlich zum Haupttastenfeld gibt es noch rechts auf dem Gerät die Zusatztasten "SELECT", "OPTION" und "START".

Zur Abfrage der Tastatur wurde keiner der üblichen Tastaturabfragebausteine verwendet. Statt dessen übernimmt ein Baustein im Atari (der sogenannte POKEY), der ansonsten noch für die serielle Schnittstelle und für die Tonerzeugung zuständig ist, die Abfrage der Tastatur. Das Ergebnis, das heißt, welche Taste gedrückt ist, stellt er in einem Register zur Verfügung. Die Tastatur, die wie bei anderen

Systemen als Matrix aufgebaut ist, wird also nicht, wie bei anderen Hobbycomputersystemen, von der CPU zeilenweise abgefragt. Dies spart Rechenzeit der CPU. Der POKEY ist in der Lage, in dem Moment, in dem eine Taste gedrückt wird, einen Interrupt bei der CPU auszulösen (den IRQ). Es ist damit möglich, die Tastatur per Interrupt zu betreiben.

Die Zusatztasten ("SELECT", "OPTION" und "START"), die sogenannten Konsolentaster, werden über einen weiteren hochintegrierten Baustein im Atari (den GTIA) abgefragt.

Die Standardausgabe des Atari erfolgt auf einen Bildschirm. Als Bildschirm eignet sich praktisch jeder Fernseher. Ein Modulator ist in die Atari-Geräte eingebaut, es wird also nur der normale Antennen- und kein Videoeingang beim Fernseher benötigt. Zur Darstellung steht auf einem Farbfernseher eine Palette von 128 Farben zur Verfügung. Dabei wird vom Betriebssystem eine Auflösung von bis zu 320 x 192 Punkten oder 24 Zeilen mit je 40 Zeichen unterstützt. Die höchste Auflösung liegt bei 384 Punkten horizontal und etwa 210 Punkten vertikal (je nach Bildschirm).

Die gesamte Bilderstellung wird von einem Baustein, der nur von Atari verwendet wird, übernommen. Dieser Baustein wird als " A N T I C " bezeichnet. ANTIC steht wohl für "Alphanumeric Television Interface Controller". Der ANTIC ist ein zweiter Mikroprozessor, der über einen eigenen Befehlssatz verfügt. Durch das Programm des ANTICs, das man an beliebiger Stelle im Speicherraum ablegen kann, wird das darzustellende Bild definiert. Der ANTIC verfügt über 14 verschiedene Modi zur Bilddarstellung. Diese Modi lassen sich beliebig auf dem Bildschirm kombinieren. Es stehen sowohl Modi zur Schriftdarstellung (auch mit Unterlängen) und Modi zur allgemeinen Zeichendarstellung (zum Beispiel für Spielfiguren) als auch zahlreiche Betriebsarten zur Einzelpunktdarstellung mit unterschiedlich hohen Auflösungen zur Verfügung.

Die Breite des Bildes läßt sich auf drei verschiedene Werte einstellen. Je nach Einstellung ergeben sich maximale hori-

zontale Auflösungen von 256, 320 oder 384 Bildpunkten pro Zeile. In der Bildumrandung erscheint eine (programmierbare) Farbe.

Der ANTIC liest alle Daten, die er zur Bilddarstellung benötigt, direkt ohne Umweg über die CPU aus dem Speicher. Diese Art des Zugriffs bezeichnet man als DMA (Direct Memory Access - direkten Speicherzugriff). Während eines Zugriffes des ANTICs auf den Speicher im DMA wird die CPU angehalten. Dadurch vermindert sich die effektive Arbeitsgeschwindigkeit der CPU. In den meisten Modi ist der Befehlsdurchsatz der CPU jedoch trotzdem höher als bei 6502-Systemen mit einem Systemtakt von einem Megahertz.

Der ANTIC erstellt nicht selbst die Farbinformationen im Videosignal, das über den Modulator auf dem Fernseher dargestellt wird. Der ANTIC überträgt die Bildinformationen, die er per DMA aus dem Speicher gelesen hat, zu einem weiteren Atari-Baustein. Dieser Baustein wird " G T I A " genannt. GTIA steht wohl für "Graphic Television Interface Adaptor". Der GTIA enthält unter anderem neun Farbregister, mit denen der Benutzer die Farben, die auf dem Bildschirm erscheinen sollen, aus einer Palette von 128 Farben auswählen kann. Der GTIA erhält vom ANTIC jeweils Informationen, welche Farbe, das heißt, die Farbe welchen Farbregisters, er abbilden soll. Durch dieses Konzept ergibt sich eine Farbpalette, die sonst meist nur von professionellen CAD-Systemen erreicht wird und sich kaum mit der anderer Mikrocomputer vergleichen läßt.

Besonders interessant dürfte auch sein, daß die Hardware des ANTICs die Verschiebung des Bildes um einzelne Punkte sowohl horizontal als auch vertikal unterstützt. Der Bildspeicher kann ohne "Tricks" an beliebiger Stelle im Speicherraum liegen. Der Zeichengenerator kann sich prinzipiell auch an jeder Stelle des Speichers befinden, muß allerdings, je nach ANTIC-Modus (je nachdem ob er 512 oder 1024 Bytes groß ist), an einer 512-Byte oder 1 KByte-Grenze des Speichers beginnen.

Mit diesem Gesamtkonzept der Bildschirmdarstellung ist der Atari leistungsfähiger als die meisten anderen Systeme dieser Klasse. Gerade für seine Zielgruppe, das heißt für Anwender im Spiel-, Hobby- und Heimbereich, dürften diese Leistungsmerkmale besonders interessant sein. Vor allem das Konzept eines zweiten Mikroprozessors im System, das Konzept des ANTICs, ermöglicht eine auf andere Art nicht zu erreichende Vielfalt von unterschiedlichen Arten der Bildschirmdarstellung. Auch bringt das Prinzip eines relativ hohen Systemtakts (1,77 MHz), bei dem der Befehlsdurchsatz der CPU durch Speicherzugriffe per DMA des ANTICs verringert wird, eine relativ hohe Verarbeitungsgeschwindigkeit. Zudem kann man während der Bearbeitung rechenintensiver Routinen einen ANTIC-Darstellungsmodus wählen, der nur wenige Zugriffe im DMA benötigt, oder kann sogar den Bildschirm abschalten. Dadurch erhöht sich der Befehlsdurchsatz der CPU erheblich.

Der Atari besitzt außerdem sehr vielfältige Möglichkeiten einer Tonausgabe. Diese wird von dem Baustein übernommen, der außerdem für die Abfrage der Tastatur zuständig ist, dem " P O K E Y ". POKEY steht wohl für "POtentiometer and KEYboard Integrated Circuit". Der Pokey verfügt über vier Tongeneratoren, die sowohl unabhängig voneinander, als auch miteinander verbunden arbeiten können. Außerdem sind für jeden Kanal vielfältige Möglichkeiten zur Verzerrung usw. vorhanden. Auch sogenannte Höhenfilter lassen sich programmieren. Mit dem POKEY kann man, entsprechende Programme vorausgesetzt, die Klangvielfalt eines Synthesizers erreichen. Es ließe sich zum Beispiel auch eine Synthesizer-Tastatur an den Atari anschließen. Der Ton des POKEYs wird über den Modulator zum Fernseher übertragen. Zur Tonausgabe wird also der normale Fernseherlautsprecher verwendet. Es ist zudem möglich, über einen Eingang des Atari, ein externes Tonsignal in das Signal des POKEYs zu mischen.

Als großer Vorteil muß die Tatsache angesehen werden, daß es möglich ist, aus dem BASIC heraus Töne zu erzeugen, ohne, wie es bei Konkurrenzmodellen nötig ist, den "POKE"-Befehl zu benutzen. Die direkte Programmierung der Tonkanäle aus Assemblerprogrammen ist etwas komplizierter als bei ver-

gleichbaren Geräten. Dafür sind die Möglichkeiten der Tonerzeugung, die der Atari bietet, aber den Möglichkeiten der Tonerzeugung bei Konkurrenzmodellen mindestens ebenbürtig. Unterstützt wird die Tonerzeugung zudem durch mehrere automatische Zähler, die in dem Moment, in dem sie bei Null angelangt sind, einen Interrupt der CPU auslösen. In der Interruptroutine kann man dann die Frequenz oder Lautstärke oder auch andere Parameter der Tonerzeugung verändern.

Die neuen Atari-Geräte (600XL/800XL) entsprechen in den meisten Details den alten Modellen (400/800). So sind alle Adressen der Hardware gleich geblieben. Basicprogramme lassen sich meist direkt austauschen oder müssen nur geringfügig modifiziert werden.

Bei den alten Atari-Modellen sind 4 Joystick-Ports vorhanden. An jeden der Joystick-Ports lassen sich, an Stelle eines Joysticks, auch zwei Paddles ("Drehregler") oder ein sogenannter Lightpen anschließen. Wenn man einen Lightpen auf eine Stele des Bildschirms hält, ist es möglich, die Position des Lightpens zu bestimmen. Dies wird durch den ANTIC ermöglicht. Die Abfrage der "Joystick-Richtungen" erfolgt über einen weiteren Baustein im Atari, die "P I A". Pia steht für "Peripheral Interface Adaptor". Die "Feuertasten" der Joysticks werden über die sogenannten Triggereingänge des GTIA abgefragt. Der analoge Wert der Stellung der Drehregler (Paddles) wird im POKEY in einen digitalen Wert umgewandelt.

Bei den neuen Geräten sind nur noch zwei Anschlüsse für Joysticks vorhanden. Dies stellt nur einen geringfügigen Nachteil dar, da es für die alten Geräte nur wenige Spiele und Anwendungen gab, die alle vier Joystickports belegten, und außerdem die Werte der Joystickports 3 und 4 simuliert werden. Die so frei gewordenen Anschlüsse der PIA wurden zum Teil intern im Atari für die sogenannte MMU (Memory Management Unit) verwendet.

Im Gegensatz zu den alten Geräten, hat Atari den Speicherraum des Prozessors bei den 600XL und 800XL-Modellen zum Teil doppelt belegt. So läßt sich zum Beispiel beim Atari 800XL (oder bei einem Atari 600XL der auf 64 KByte RAM nachgerüstet wurde) das Betriebssystem abschalten. An Stelle des Betriebssystems steht dann RAM zur Verfügung. So ist es zum Beispiel möglich, das Betriebssystem der Atari 400 und 800-Modelle in die neuen Geräte zu laden. Außerdem verfügen die neuen Modelle über ein Selbsttestprogramm (im ROM), das in den normalen Arbeitsspeicherbereich eingeblendet wird, wenn beim Einschalten des Atari die "OPTION"-Taste gedrückt ist und kein Cartridge eingesteckt ist. Mit diesem Selbsttestprogramm ist es möglich den RAM-Speicher, die Tonerzeugung, die Tastatur und den Bildschirm zu testen.

Die verschiedenen Speicherbelegungen wählt man mit mehreren Portleitungen der PIA, die bei den alten Geräten für den dritten und vierten Joystickport verwendet wurden. Die sogenannte MMU gibt, je nach Einstellung und "Situation", Freigaben an die einzelnen Hardwarekomponenten des Atari. Bei der MMU handelt es sich um einen Logikbaustein, einen sogenannten PAL-Baustein (PAL steht hier für "Programmable Array Logic" und nicht für das Fernseherfarbsystem), der speziell für Atari programmiert ist. In seiner Funktion ist dieser PAL-Baustein mit gewöhnlichen TTL-Bausteinen vergleichbar.

Um andere Programme ohne Ladezeiten der Diskette oder Cassette verwenden zu können, verfügt der Atari über einen Cartridge-Slot. In diesen Cartridge-Slot lassen sich Module einstecken, die bis zu 16 KByte im Speicher belegen können. Diese Module bieten eine höhere Datensicherheit als Cassetten oder Disketten, unter anderem durch eine grundsätzlich kaum begrenzte Lebensdauer. Wenn ein Cartridge in den Slot gesteckt ist, wird der vom Speicher im Cartridge überdeckte RAM-Speicher (in zwei 8 KByte-Blöcken) abgeschaltet. Außerdem startet das Betriebssystem automatisch das im Cartridge enthaltene Programm. Zudem ist im Cartridge-Slot noch eine Freigabe-Leitung für einen 256 Byte-Block (eine Page) im Bereich der anderen I/O-Bausteine vorhanden. Damit ist es möglich, im Cartridge verschiedene ROM-Bereiche umzuschal-

ten. Man kann also auch ROMs mit mehr als 16 KByte verwenden. Selbstverständlich könnte man mit Hilfe dieser Freigabe-Leitung auch eine Karte mit einem Peripheriebaustein im Slot betreiben. Dies ist aber bei den Atari 400 und 800-Modellen umständlich, da für den Betrieb die Klappe über dem Slot geschlossen sein muß. Bei den neuen Geräten hat Atari auf die Klappe wohl aus Kostengründen verzichtet. Dabei hat sich aber die Bildqualität geringfügig verschlechtert (da die Abschirmung nicht mehr so gut ist). Dieser Effekt ist aber von Gerät zu Gerät unterschiedlich. Es kommt vor allem auch darauf an, welche anderen Geräte in der Nähe des Computers arbeiten.

Ursprünglich enthielten die Atari-Computer nur das Betriebssystem. Das BASIC, das es als Cartridge gab, mußte extra erworben werden. Atari ging offenbar davon aus, daß nicht jeder das BASIC haben wollte. Später gehörte ein BASIC-Cartridge zum Lieferumfang dazu. Jeder, der einen Atari-Computer (400 oder 800) kaufte, bekam einen BASIC-Cartridge.

Die neuen Atari-Computer (600XL/800XL) enthalten das BASIC bereits fest eingebaut. Es ist vollkommen kompatibel zu den Cartridge-Versionen des BASICs. Die Programme lassen sich also direkt austauschen, sofern nicht einige Betriebssystemroutinen, bestimmte Betriebssystemadressen oder bestimmte "Zero-Page-Adressen" verwendet worden sind. Auch ist es möglich, mit Hilfe der MMU das BASIC abzuschalten. Wenn ein Cartridge in den Slot gesteckt wird, wird das eingebaute BASIC abgeschaltet, sofern der Cartridge den gleichen Speicherraum wie das BASIC belegt. So ist es möglich, sowohl andere Programmiersprachen, als auch Erweiterungen des BASICs als Cartridge zu verwenden.

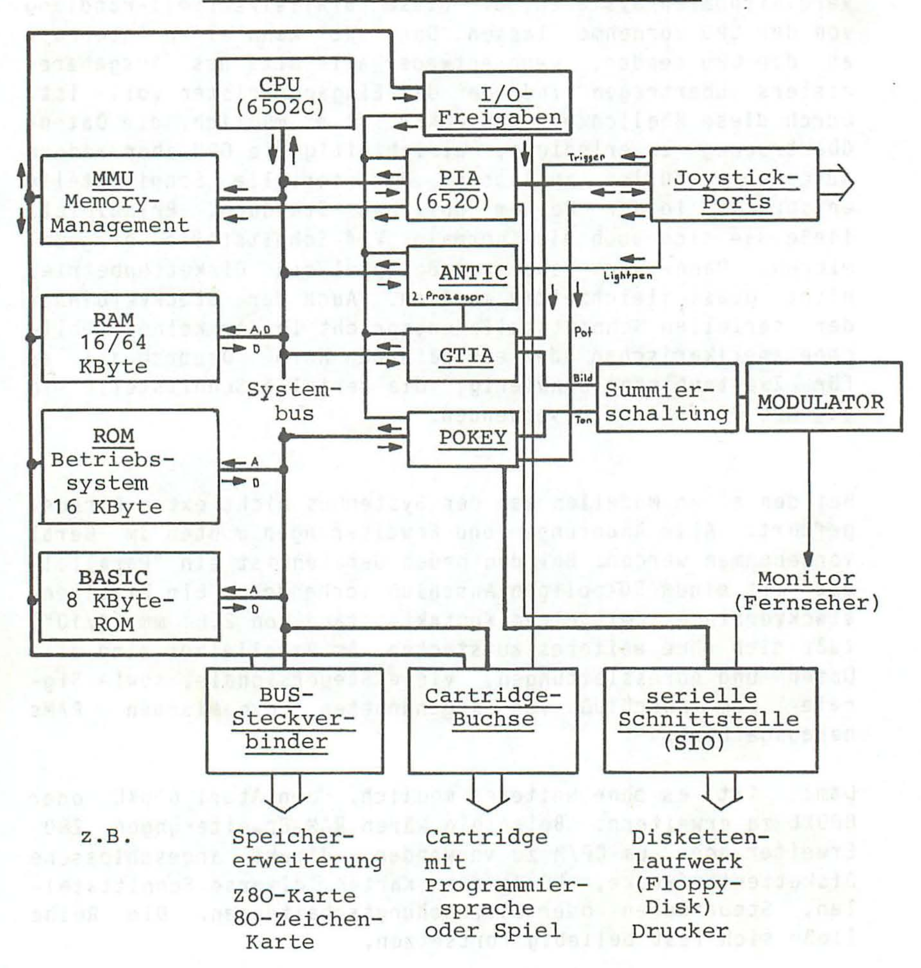
Diskettenlaufwerke, Drucker, der Cassettenrecorder sowie andere Peripheriegeräte werden an eine serielle Schnittstelle angeschlossen. Dieser Übertragungskanal wird vom POKEY bedient. Mit ihr sind Übertragungsraten bis zu 19200 Baud (Bit/Sekunde) möglich. Die Wandlung der parallen Daten in serielle und umgekehrt wird von der POKEY-Hardware über-

nommen. So ergeben sich höhere Übertragungsraten als bei vergleichbaren Systemen, die diese parallel/seriell-Wandlung von der CPU vornehmen lassen. Der POKEY kann einen Interrupt an die CPU senden, wenn entweder alle Bits des Ausgaberegisters übertragen sind oder das Eingaberegister voll ist. Durch diese Möglichkeit des POKEY ist es möglich, die Datenübertragung zu erledigen, gleichzeitig die CPU aber andere Aufgaben erfüllen zu lassen. Die serielle Schnittstelle entspricht leider keinem üblichen Standard. Prinzipiell ließe sie sich auch als "normale" V24-Schnitstelle programmieren. Dann wäre aber zum Beispiel ein Diskettenbetrieb nicht quasi-gleichzeitig möglich. Auch der Steckverbinder der seriellen Schnittstelle entspricht leider keiner üblichen amerikanischen oder europäischen Norm. Dadurch ist es für Zweitanbieter schwierig, die serielle Schnitstelle in eigenen Produkten zu verwenden.

Bei den alten Modellen war der Systembus nicht extra herausgeführt. Alle Änderungen und Erweiterungen mußten im Gerät vorgenommen werden. Bei den neuen Geräten ist ein "Parallelbus" mit einem 50-poligen Anschluß vorhanden. Ein Platinensteckverbinder mit einem Kontaktabstand von 2,54 mm (1/10") läßt sich ohne weiteres aufstecken. Am Parallelbus sind alle Daten- und Adressleitungen, viele Steuersignale, sowie Signale zum Anschluß von sogenanneten dynamischen RAMs herausgeführt.

Damit ist es ohne weiteres möglich, den Atari 600XL oder 800XL zu erweitern. Beispiele wären RAM-Erweiterungen, Z80-Erweiterungen um CP/M zu verwenden, direkt angeschlossene Diskettenlaufwerke, 80-Zeichen-Karten, diverse Schnittstellen, Steuerungen oder Überwachungsschaltungen. Die Reihe ließe sich fast beliebig fortsetzen.

Das Blockschaltbild der Atari-Geräte



(Die Stromversorgung wurde nicht berücksichtigt !)

Die Atari 400 und 800-Geräte verfügten in mehreren Bausteinen über 10 KByte ROM. Darin befand sich im wesentlichen das Betriebssystem und ein Zeichengenerator. Bei den neuen Geräten setzte Atari ein modernes 16 KByte-ROM ein. Der so hinzugekommene ROM-Speicherraum wurde unter anderem für die Selbsttestroutine, einen zweiten Zeichengenerator und ein erweitertes Betriebssytem verwendet.

Atari hat schon bei den alten Geräten ein "fast" echtes Betriebssystem verwendet. Im Gegensatz zu vielen anderen sogenannten Betriebssystemen besteht das Atari-Betriebssystem nicht nur aus elementaren Betriebsroutinen, sondern verfügt über leistungsfähige Betriebssystemroutinen, die universell verwendet werden können. Zum Beispiel wird der Verkehr mit Peripheriegeräten durch eine universelle Prozedur erledigt. Das aufrufende Programm muß der I/O-Routine die Ziel- bzw. Quellenangaben und alle anderen Parameter in einem sogenannten I/O-Controlblock übergeben. Dieses Konzept findet man sonst erst bei diskettenorientierten Betriebssystemen wie CP/M, MSDOS, UNIX oder QNX. Bei den Betriebssystemen der anderen Hobbycomputer, ist gewöhnlich für jedes Ein- bzw. Ausgabegerät (Bildschirm, Tastatur, Diskette, Cassette, Drucker usw.) eine eigene Routine vorhanden. Jede dieser Routinen erwartet die Daten meist in einem anderen Format. Wenn man nun ein Programm schreiben möchte, das die Ausgaben entweder in einer Diskettendatei oder auf einem Drucker bringt, stellen sich bei derartigen Betriebssystemen meist nicht ganz unerhebliche Probleme.

Die Ein- und Ausgabe des Atari ist grundsätzlich, von ihrer Hardware her, interruptfähig. Dies wird vom Betriebssystem voll unterstützt. In eigenen Maschinenprogrammen ist es also ohne weiteres möglich, zu drucken oder Ein- und Ausgaben der Diskettenlaufwerke auszuführen, während der Prozessor ansonsten noch oder bereits mit einer anderen Aufgabe beschäftigt ist. Diese Möglichkeiten des Betriebssytems werden vom BASIC jedoch nicht ausgenutzt, da dies bei einem BASIC auf einem Einplatzsystem ohne Multitask-Option ohnehin kaum sinnvoll ist. Es ließe sich aber zum Beispiel ein Textsystem

programmieren, bei dem der Benutzer praktisch ohne Rücksicht auf Diskettenschreib- oder -leseoperationen oder Such- und Austauschfunktionen des Textsystems, dauermäßig Eingaben machen kann. Der Datenverkehr mit den Disketten wird über das Betriebssystem im Interrupt vorgenommen. Um keine Tastatureingaben verlorengehen zu lassen, müßte der POKEY immer, wenn eine Taste gedrückt wird, einen Interrupt auslösen. In der Unterbrechungsroutine würden alle Tastatureingaben in einen Tastaturpuffer geladen. Der "Vordergrundjob" (die Tätigkeit der CPU, die von den Interrupts unterbrochen wird, das eigentliche Textverarbeitungsprogramm) holt sich die Eingaben bei Bedarf aus dem Tastaturpuffer. Nach diesem "Muster" lassen sich noch viele andere Programme entwerfen.

Der normale Betrieb des Atari wird alle 1/50 Sekunde unterbrochen. Während der Vertikalsynchronisation des Bildschirms wird ein Interrupt ausgelöst. In der dazugehörigen Interruptroutine werden mehrere Aufgaben erfüllt. So werden mehrere Speicherzellen erhöht oder erniedrigt, diese Speicherstellen stellen dadurch Softwarezähler dar, mit denen es möglich ist, den Zeitablauf der Programme (insbesondere bei Spielen wichtig !) zu steuern. Mehrere Register lassen sich, entsprechend umgerechnet, als Uhr verwenden. Eine weitere Hauptaufgabe dieser "Vertikal-Interrupt-Routine" ist es, sogenannte Schattenregister zur Verfügung zu stellen. Da viele Hardwareregister nur schreibend oder nur lesend sind, werden vom Betriebssystem Schattenregister im RAM erzeugt. Die Werte der Schattenregister bzw. der Hardwareregister werden so 50 mal pro Sekunde aktualisiert. Die Schattenregister kann man im Gegensatz zu den meisten Hardwareregistern auch zum Beispiel durch "Increment"- und "Decrement"-Befehle modifizieren. Allerdings muß man dabei immer beachten, daß die Inhalte der Schattenregister nicht unbedingt aktuell sind.

Das Betriebssystem der Atari-Geräte ist grundsätzlich sehr sauber programmiert. Die berühmte "Hauptsache: funktioniert"-Mentalität, die man bei vielen anderen Geräten dieser Preisklasse vorfindet, wird man bei Atari, von sehr wenigen

Ausnahmen abgesehen, vergeblich suchen. Das Atari-Betriebssystem verfügt über relativ wenige, dafür aber universelle Unterprogramme. Alle wichtigen Routinen werden, auch im Betriebssystem intern, über eine sogenannte Vektortabelle angesprungen. In dieser Vektortabelle stehen die Startadressen der einzelnen Routinen. Durch dieses Konzept ist es relativ einfach, das Betriebssystem sowohl aus Maschinenprogrammen heraus zu benutzen, als auch es zu verändern.

Bei den neuen Geräten ist das Betriebssystem teilweise verändert und verbessert worden. So stehen beim neuen Betriebssystem einige neue Routinen zur Verfügung. Die Vektortabelle ist prinzipiell nicht verändert worden, es sind lediglich einige Vektoren angefügt worden. Auch mußte natürlich die veränderte Hardware (zum Beispiel die MMU und das eingebaute BASIC) im Betriebssystem berücksichtigt werden. Dies hat zur Folge, daß sehr viele Routinen im neuen Betriebssystem an anderer Stelle geringfügig modifiziert vorgefunden werden. Auch viele Speicherstellen im RAM, die vom Betriebssystem als Zwischenspeicher benutzt werden, liegen jetzt an anderen Stellen. Dies hat zur Folge, daß nicht mehr alle alten Programme auf den neuen Geräten laufen. Programme, die aber Betriebssystemroutinen nur über die Vektortabelle anspringen und sich auch ansonsten an die "Betriebssystemspezifikationen" halten, werden meist ohne Problem laufen. Programme, die "Tricks" anwenden, laufen dagegen oft nur nach einer, von Fall zu Fall unterschiedlich schwierigen Anpassung auf den neuen Geräten.

D I E   H A R D W A R E   D E S   A T A R I

1) ALLGEMEINES

2) ANSCHLUSSPLÄNE UND SPEICHERAUFTEILUNG

Es erstaunt nicht weiter, daß beim Atari 600XL/800XL neben der allgemein guten Konzeption auch die Hardware einen recht aufgeräumten und durchdachten Eindruck macht, wobei klar sein dürfte, daß in dieser Preisklasse nicht jedem alles recht gemacht werden kann.

Einen Großteil der Strukturierung der Baugruppen wird im 600XL/800XL, wie schon in den alten 400/800-er Modellen, von den hochintegrierten Spezialbausteinen ANTIC, POKEY und GTIA, in Verbindung mit den Standardbausteinen CPU und PIA übernommen.

Als Speicher werden in den beiden neuen Geräten ein 16KByte großes ROM für das Betriebssystem und ein 8KByte großes ROM für das eingebaute BASIC verwendet.

Der einzige große Unterschied zwischen 600XL und 800XL ist bekanntlich, daß der 800XL einen auf 64KByte erweiterten RAM-Bereich besitzt. Diese Erweiterung besteht nicht aus weiteren eingesteckten Bausteinen, sondern aus einer etwas anderen Konzeption.

Während der 600XL für seine 16KByte RAM zwei Bausteine verwendet, die jeweils 16.384 4bit-Worte enthalten, ist der 800XL mit 8 wesentlich gebräuchlicheren und damit preiswerteren Bausteinen ausgerüstet, die jeweils 65.536 1bit-Worte enthalten.

Schon alleine bei der Stromversorgung bemerkt man, daß die Atari-Computer keine 'Wird schon funktionieren'-Ware sind.

Die vom extern geregelten Netzteil kommende Spannung wird gleich zu Beginn nach dem Hauptschalter in drei Pfade aufgeteilt, die alle gegeneinander durch Induktivitäten und Block-Kondensatoren entkoppelt sind.

Ein Pfad ist hauptsächlich für die hochintegrierten Bausteine und die direkt daran hängenden Pufferbausteine zuständig, der zweite Pfad führt zu den restlichen Logikbausteinen und den RAMs und der dritte Stromversorgungsteil ist ausschließlich für den HF-Modulator vorgesehen. Es ist auch

der induktiv am besten entkoppelte Zweig, was Sörungen sowohl der Digitalsignale durch die Hochfrequenz als auch umgekehrt verhindern soll.

Das Herz des Computers ist dessen Taktgenerator. Dieser hier erzeugt eine quarzgenaue 3.546894 MHz-Schwingung, die direkt für die Farbtakterzeugung herangezogen wird.

Die Hälfte dieser Grundfrequenz (1.773477MHz) wird als Arbeitstakt für die CPU benutzt und steuert damit zeitlich alle nicht direkt zur Farbansteuerung gehörenden Funktionen.

Ein von der CPU geliefertes Taktsignal wird benutzt, um über eine sogenannte Spannungs-Verdopplerschaltung eine Referenzspannung für CADJ des GTIA zu erzeugen. Dies ist bei diesen Geräten notwendig, da beim alten 400/800 eine Versorgungsspannung von 12V existierte, die aufgrund der modernen RAM-Bausteine hier keine Verwendung mehr fände.

Um das System in einen geordneten Zustand bringen zu können, benötigt es einen Reset-Impuls. Dieser wird automatisch beim Einschalten durch ein RC-Glied erzeugt. Der dabei gegen Masse liegende Kondensator kann über eine externe Leitung entladen werden; diese externe Leitung führt zur RESET-Taste der Konsole. Es werden von diesem Signal die Bausteine ANTIC, PIA und CPU zurückgesetzt. Die anderen Bausteine benötigen dieses Signal nicht, sie werden softwaremäßig initialisiert.

Die CPU steuert über ihre Adreß-, Daten- und Kontrollbusleitungen die anderen hochintegrierten Schaltkreise an. Sie läßt sich jedoch auch über die vom ANTIC stammende HALT-Leitung anhalten, so daß sie ihre Leitungen auf einen hochohmigen Pegel setzt. Das hat zur Folge, daß die Busleitungen jetzt frei sind, damit zum Beispiel der ANTIC seinen direkten Speicherzugriff (DMA) machen kann.

Sämtliche Speicher (also RAM wie ROM) werden über einen weiteren, weniger hoch integrierten Baustein angesteuert.

Dieses IC ist ein programmierbarer Logikbaustein (PAL) und erfüllt unter anderem die Aufgaben einer Speicher-Dekodier-Logik, die sonst mit diversen Standard-Dekodern hätte aufgebaut werden müssen. Da sie jedoch nicht nur Dekodier-, sonden auch Speicherblock-Verschiebeaufgaben besitzt, wäre die Bezeichnung 'Memory-Decoder' ihr nicht gerecht geworden.

Da dieses IC in gewissem Rahmen auch eine Verwaltungslogik beinhaltet, ist der Ausdruck Memory-Management-Unit (MMU) wohl der bestgeeignete.

Bevor die einzelnen Ein- und Ausgänge diskutiert werden sollen, noch eine kleine Bemerkung zur Syntax der Leitungsbezeichner: Hinter jedem Signal steht hinter einem Doppelpunkt der Pegel, bei dem das Signal aktiv ist. Ist also zum Beispiel Signal XYZ bei Low-Pegel aktiv, so heißt das Signal XYZ:L, sonst XYZ:H.

Als Eingänge besitzt die MMU zum Beispiel die fünf obersten Adressbit und die vom ANTIC stammende REFresh-Information, die besagt, daß es sich bei dem folgenden Speicherzugriff um einen Refresh-Zyklus handelt (die verwendeten RAM-Bausteine müssen in bestimmten Zeitabständen angesprochen werden, sonst verlieren sie ihre Inhalte. Siehe allgemeine Datenblätter dynamischer RAMs).

Mit diesen Grundinformationen ist die MMU schon in der Lage, den gesamten 64KByte-Speicherbereich in 32 2Kbyte große Blöcke aufzuteilen. Die Blöcke würden prinzipiell gehen von

```
Block 0   $0000 bis $07ff
Block 1   $0800 bis $0fff
Block 2   $1000 bis $17ff
  .         .         .
  .         .         .
  .         .         .
Block 30  $f000 bis $f7ff
Block 31  $f800 bis $ffff
```

Die MMU faßt jedoch größere Blcokgruppen jeweils zu Einheiten zusammen.

So ist zum Beispiel der Bereich von $0000 bis $4fff (die untersten 16KByte) immer eine Einheit, die sowohl beim 600XL als auch beim 800XL aus RAM besteht. Dann kommt ein Block, der besondere Beachtung verdient, um dann von $5800 bis $7fff wieder einen normalen RAM-Speicherbereich anzusprechen, egal, ob dort physikalische RAM-Bausteine liegen oder nicht.

Der Block Nummer 10 ($5000 bis $57ff) hat, wie gesagt, einen speziellen Status. Der Bereich ist normalerweise für den Einsatz von RAM vorgesehen, es läßt sich jedoch auch in diesem Block abschalten. Dazu führt von PIA-PORT B7 eine Leitung zum MAP-Eingang der MMU. Ist dieser Eingang auf 1, so bleibt wie gewohnt das RAM eingeschaltet. Geht er jedoch auf Null, so wird RAM-Block 10 abgeschaltet und der hinter den I/O-Bausteinen in Block 26 ($d000 bis $d7ff) versteckt liegende ROM-Bereich wird hier hinuntergespiegelt. Es handelt sich bei diesen 2KByte Programm um die Selbsttest-Routine, die eingeschaltet wird, wenn beim Einschalten des Computers die OPTION-Taste gedrückt wird und keine Diskette angeschlossen ist.

Aber noch zwei andere Umschalteleitungen führen von Port B zur MMU: PORT B0 führt direkt zu ROM:H/RAM:L an der MMU und bewirkt dort, daß beim Nullwerden der Leitung das im Bereich $c000 bis $cfff und $d800 bis $ffff liegende Betriebssystem-ROM ab- und das eventuell dahinter liegende RAM eingeschaltet wird. Bei Benutzung dieser Adressen ist unbedingt darauf zu achten, daß das System abstürzt, wenn kein RAM an dieser Stelle vorhanden ist oder kein vernünftiges System darin steht.

Um sicherzustellen, daß beim Kaltstart immer das ROM aktiviert ist, geht man davon aus, daß solch ein PIA-Pin nach einem RESET-Impuls automatisch als Eingang programmiert wird. Es muß dann nur noch der nun offene Eingang über einen Pull-up-Widerstand hochgezogen werden und schon ist das ROM eingeschaltet.

Die dritte und letzte Verbindung zwischen MMU und PIA wird durch PORT B1 festgelegt, dessen Signal direkt an BasE:L geht. Ist diese Leitung auf Null, so ist der 8 KByte große Bereich $a000 bis $bfff mit dem internen BASIC-ROM belegt, ansonsten ist dieser Bereich frei für anderes.

Dieses 'andere' ist im Normalfall RAM, das an diese Stelle eingeblendet werden kann unter der Voraussetzung, daß es sich um einen 800XL oder 600XL mit 64KByte-Karte handelt.

Die MMU hat jedoch noch zwei weitere Eingangsleitungen, mit denen sie Speicherbereiche umschalten kann; sie dienen der hardwaremäßigen Verarbeitung von Cartridges, also Steckmodulen.

Liegt ein Steckmodul im Speicherbereich von $a000 bis $bfff (zum Beispiel das alte BASIC-Modul, das jetzt eingebaut ist oder der Atari-Assembler oder Spiele wie STARRAIDER u.ä.), so wird dies der MMU über die Leitung EN5:H mitgeteilt, liegt das ROM-Modul (auch) im Bereich $8000 bis $9fff, so wird (auch) die Leitung EN4:H auf High gesetzt. Dadurch 'weiß' die MMU entsprechend, daß sie das dahinterliegende RAM nicht aktivieren darf.

Als letzte Eingangsleitung ist noch die MPD:L Leitung zu nennen. Sie ist eine externe Schaltleitung und bewirkt, daß aus dem Betriebssystembereich der auf die I/O-Gruppen folgende Bereich $d800 bis $dfff (Mathematikroutinen-ROM, deshalb auch Math-Pack-Disable-Leitung) abgeschaltet wird. Dies wird für spätere Hardwareergänzungen verwendet.

Neben diesen ganzen Eingangsleitungen verfügt die MMU selbstverständlich auch über Ausgangsleitungen, die alle als Selektierungsleitungen dienen.

Ist EN4:H oder EN5:H High, so wird beim Ansprechen des entsprechenden Adressbereichs (also $8000 bis $9fff respektive $a000 bis $bfff) die Leitung SEL4:L beziehungsweise SEL5:L auf Null gezogen, um das entsprechende ROM-Modul zu aktivieren.

Das Betriebssystem-ROM wird durch die Leitung OS:L nach folgenden Kriterien selektiert:

MPD:L	ROM:H/RAM:L	Adressbereich	Ausgang OS:L
H	H	$c000 bis $cfff	L
H	H	$d800 bis $ffff	L
x	L	generell inaktiv	H
L	H	$c000 bis $cfff	L
L	H	$d800 bis $dfff	H
L	H	$e000 bis $ffff	L

Die Leitung I/O:L bewirkt beim Ansprechen des Bereichs $d000 bis $d7ff, daß der Memory-map-I/O angesprochen wird. Memory-map heißt hier, daß für die Ein-/Ausgabebausteine kein separater Adreßraum zur Verfügung gestellt wird, sondern ein Teilbereich des Speicherraums dafür abgezweigt wird.

RAM-Speicher kann dort, wo es notwendig ist, über die Leitung CasInh:L von der MMU abgeschaltet werden. Diese Ausgangsleitung bewirkt bei entsprechender Decodierung beim RAM, daß das sogenannte Column Address Strobe, was die letztendliche Selektion einer dynamischen Speicherstelle bewirkt, unterdrückt wird. Dies wird deshalb auf diesem Wege vollzogen, weil sichergestellt werden muß, daß trotz aller Deselektion immer noch der Refresh zu den RAM-Bausteinen durchkommt.

Um von der MMU gleich zu den RAM-Bausteinen weiterzugehen, bedarf es auf elektronischem Weg einiger Timing-Glieder.

Allgemein kann man sagen, daß die Ansteuerung dynamischer Speicher nur wegen der genau einzuhaltenden zeitlichen Folge der von den Speicherbausteinen verlangten Signale extrem kompliziert ist.

Beim 600XL/800XL werden diese Zeitprobleme nun durch spezielle Zeitglieder gelöst, die eingehende Signale um wenigstens 25, höchstes ca. 260 milliardstel Sekunden verzögern.

Auf die gesamte Decoderlogik wirken zwei Selektionssignale:

Zum einen das CasInh:L von der MMU, zum anderen ein extern anzulegendes ExtSel:L, das beim Nullwerden das Ansteuern des internen RAMs verhindert.

Neben der Speicherverwaltung gibt es noch zwei andere große Schaltungsbereiche innerhalb des Atari. Der eine wäre die Erzeugung der Farb-, Bild- und Tonsignale, zu denen nichts weiter besonderes zu sagen ist, da sie aus Standard-Hardware bestehen, wie zum Beispiel einem Digital-Analogwandler für die Luminanzwerte des GTIA oder einen 3-stufigen HF-Verstärker für das komplette FBAS-Signal. Außerdem ist es von Atari nicht vorgesehen, daß in diesen Schaltungsteil von externer Hardware eingegriffen wird.

Der zweite Bereich ist die Ein-/Ausgabe:

So existieren Multiplexer/Demultiplexer für die Tastaturansteuerung, Analog- und Digitaleingänge von POKEY und PORT A für Paddle und Joystick sowie zwei Systembusverbinder.

Die Analog- und Digitaleingänge sind über Pull-up-Widerstände und gegen Masse gelegte Kondensatoren entstört.

Im Gegensatz zu den alten 400/800-er Modellen ist zu beachten, daß die Ein-/Ausgänge der PIA nicht mehr durch Serienwiderstände gegen Überlast gesichert sind. Das hat den Nachteil, daß die Ausgänge nun beim Experimentieren leicht zerstört werden können, demgegenüber widerum den Vorteil, keine Probleme mehr zu bekommen, wenn der Ausgang mit normalen Lasten in seinem Grenzbereich betrieben werden soll.

Die Joysickeingänge und Paddle-Leitungen werden über 9-polige D-SUB-Miniaturbuchsen herausgeführt, deren Anschlußbild aus der folgenden Tabelle zu entnehmen ist:

Pin Nummer	Anschluß geht intern zu Joy1 / Joy2	Wird benutzt für
1	PORT A0 / A4	Vorwärts (Joy)
2	PORT A1 / A5	Rückwärts (Joy)
3	PORT A2 / A6	N. links (JOY), linker Auslöser beim Paddle
4	PORT A3 / A7	N. rechts (Joy), rechter Auslöser beim Paddle
5	POKEY P1 / P3	linker Potentiometerwert Paddle
6	GTIA T0 / T1	Trigger (Feuerknopf) vom Joystick oder Lightpeneingang
7	Ub +5V / +5V	Betriebsspannung normal belastbar
8	GND / GND	Signalmasse
9	POKEY P0 / P2	recht. Potentiometerwert Paddle

Ein eventueller Lightpen (Lichtgriffel, Schalter, der beim Erkennen bestimmter Helligkeit schließt) kann beim 600XL/800XL im Gegensatz zu den alten Geräten in jeder der beiden Buchsen eingesetzt werden. Beim Einsatz von zwei Lichtgriffeln gleichzeitig werden die negativen Eingangssignale mit ODER verknüpft (also die Originaleingänge mit logischem UND).

Besondere Beachtung verdienen die beiden direkten CPU-Busverbinder. Der eine ist unter dem Namen ROM-Schacht bekannt und hält neben den untersten 13 Adressleitungen (für 8KByte Adreßraum), 8 Datenleitungen, Masse und +5V Versorgungsspannung einige weitere Schaltleitungen.

Da wären zum Beispiel die bei der MMU erwähnten Leitungen Sel4:L, Sel5:L, En4:H und En5:H. Sie dienen als Selektionsleitungen beziehungsweise als Informationsleitungen für die MMU, daß an dieser Stelle jetzt ROM liegt und das RAM darunter abgeschaltet wird.

Weiter wird die I/O-Selektionsleitung I/O5:L herausgeführt. Sie wird aktiv (Low), wenn der I/O-Bereich $d500 bis $d5ff angesprochen wird.

Um den Datentransfer mit den im ROM-Schacht steckenden Bauteilen sicher gestalten zu können, werden noch die bekannten 6502-CPU-Signale R:H/W:L sowie Φ2 herausgeführt.

An dem hinten am Gerät installierten CPU-Bus liegen wesentlich mehr Signale an. Neben allen 16 Adress- und 8 Datenbits liegt an mehreren Stellen des Verbinders eine Masseverbindung und für die +5V-Versorgung sind ebenfalls zwei Leitungen vorgesehen.

Die Leitung ExtSel:L sperrt im aktiven Zustand das intern vorhandene RAM und wird hauptsächlich bei Anstecken einer RAM-Karte an den 600XL benötigt.

RESET:L ist das nichtsynchronisierte Ausgangssignal, mit dem die internen Bausteine zurückgesetzt werden.

IRQ:L ist ein Eingang, der am CPU-Pin IRQ anliegt. Es ist darauf zu achten, daß alle anderen unterbrechenden Geräte wie der POKEY und die PIA ebenfalls ihre Open-Drain-Ausgänge auf dieser Leitung zu liegen haben, und daß dieser Eingang durch einen 3kOhm-Widerstand intern bereits 'gepullt' wird.

LR:H/W:L ist ein gelatchtes, gepuffertes Signal, das dem CPU-R/W entspricht. Das Latchen geschieht zeitabhängig von Φ2 zum gleichen Zeitpunkt, an dem die Verknüpfung aus CasInh:L und ExtSel:L gelatcht wird und kann direkt zur Ansteuerung von Speicherbausteinen verwendet werden.

RDY:L ist verbunden mit dem gleichnamigen CPU-Pin, aber es ist zu beachten, daß der ANTIC über seine ebenfalls gleichnamige Open-Drain-Leitung immer Zugriff auf die CPU haben muß. Bei Verwendung dieser Leitung gilt also die gleiche Vorsicht wie bei der IRQ:L-Leitung.

CasInh:L ist, wie schon bei der MMU besprochen, Ausgangssignal zum Disablen externer dynamischer Rams. Es kann auch für statische Baugruppen verwendet werden, erfordert dann jedoch leichte vorherige Umwandlungsarbeit.

CAS:L ist das reine Timing-Signal, wann der Column-Address-Strobe beim dynamischen Speicher zu erfolgen hat, wenn nicht CasInh:L oder ExtSel:L aktiv sind.

Für RAS:L gilt ähnliches wie für CAS:L, nur daß dieses Signal auch beim Refresh-Zyklus generiert wird und nicht von CasInh:L abhängig ist.

Ref:L ist die Hardwarekopie des entsprechenden Ausgangs beim ANTIC und findet nur dann Verwendung, wenn aus den RAS/CAS-Signalen Informationen über den momentanen Busstatus gezogen werden sollen. Dies wäre zum Beispiel notwendig, wenn über RAS:L und CAS:L Baugruppen angesprochen werden sollen, die keinen Refresh benötigen. In wieweit dies sinnvoll ist und nicht einfach benötigte Signale aus dem restlichen CPU-Timing entnommen werden, bleibt dem jeweiligen Entwickler überlassen; wichtig werden diese Signale, wenn von außen auf das System zugegriffen werden soll.

MPD:L ist der über Pull-Up inaktiv gehaltene MMU-Eingang zum Disablen des Mathematikroutinen-ROMs im Bereich $d800 bis $dfff und wird bislang noch von keiner (im Handel bekannten) Hardware verwendet.

Das letzte auf dem Bus verwendbare Signal ist Audio In, über das Signale in den vom Rechner generierten Sound eingemischt werden können. Der Eingang ist über einen 4u7F-Kondensator gleichspannungsentkoppelt, kann jedoch auch unter bestimmten Voraussetzungen wie nochmaliger Verstärkung etc. als Ausgang verwendet werden. Dabei wäre dann mit einem Ausgangsspannungspegel von maximal 200mV zu rechnen.

Als Gesamtübersicht über diesen Verbinder läßt sich sagen, daß er alle Erweiterungsmöglichkeiten nach außen hin offen läßt, wobei er es unmöglich macht, etwas an der internen Hardwarestruktur zu ändern, was nicht von Atari direkt geplant war. Als weiterer wichtiger Punkt dürfte herausgekommen sein, daß es sich bei dieser Schnittstelle nicht etwa um eine Centronics-kompatible handelt, wie es leider immer wieder von unwissenden Verkäufern erzählt wird.

Allgemein sind die Signale, die nach außen geführt werden, nicht gepuffert. Es ist also davon auszugehen, daß die Daten- und Adressleitungen jeweils eine LS-TTL-Last vertragen, die anderen Signale wie RESET:L, CAS:L und andere vertragen nicht mehr als 5 weitere LS-TTL-Lasten. Es ist gerade bei CAS:L und RAS:L zu beachten, daß dynamische Speicher extrem hohe Eingangskapazitäten besitzen, die das Fan-out der Signalquellen erheblich schwächen.

```
********************************************
*                                          *
*                                          *
*   ANSCHLUSSPLÄNE UND SPEICHERAUFTEILUNG  *
*   -------------------------------------  *
*                                          *
********************************************
```

Die Speicheraufteilung des Atari

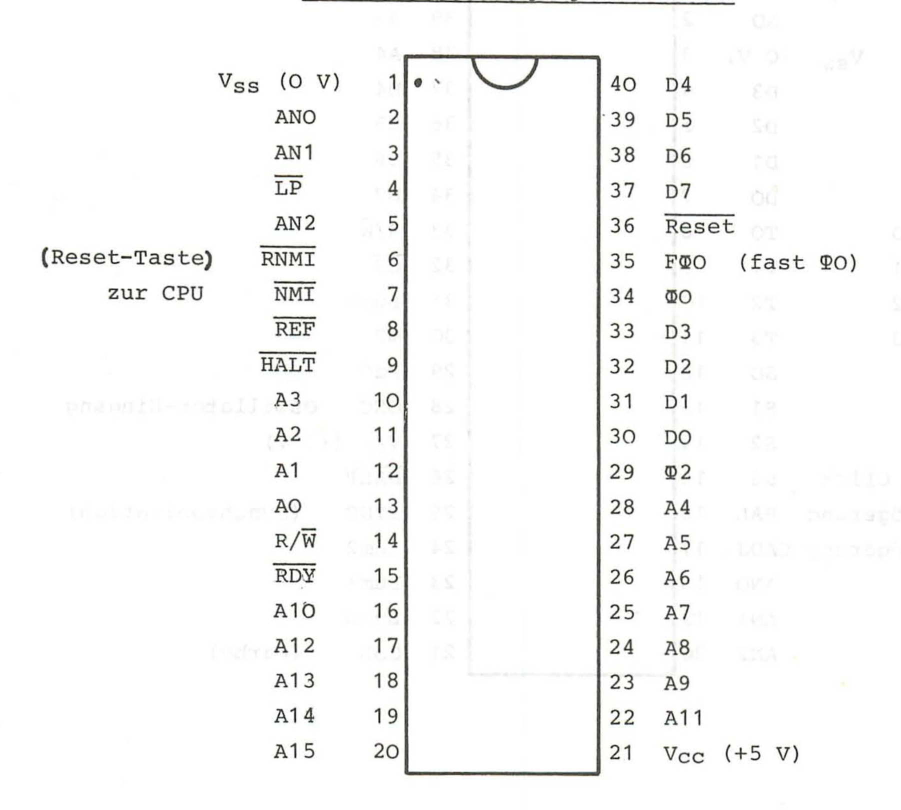
Adresse	RAM	ROM	
0000	RAM (bei allen) 16 KByte		
4000	RAM (800/800XL) 16 KByte		
5000–57FF			Spiegelung des Selbsttest-ROMs
8000	RAM (800/800XL) 8 KByte		Speicherbereich des Cartridge ($\overline{\text{Sel4}}$)
A000	RAM (800/800XL) 8 KByte	BASIC (600XL/800XL) 8 KByte	Speicherbereich des Cartridge ($\overline{\text{Sel5}}$)
C000	RAM 800XL 4 KByte	ROM (600XL/800XL) 4 KByte	
D000	RAM 800XL nicht benutzbar !	Selbsttestprogramm, nach 5000-57FF spiegelbar	GTIA, 256 Bytes
D100			unbenutzt
D200			POKEY, 256 Bytes
D300			PIA, 256 Bytes
D400			ANTIC, 256 Byte
D500			Freigabe beim Cartridge
D600			unbenutzt (512 Bytes)
D800	RAM 800XL 10 KByte	Math-Pack (im Betriebssystem-ROM, 2 KByte)	
F000		Betriebssytem (ROM, 8 KByte)	
FFFF			

Die Anschlußbelegung der CPU 6502C im Atari

Signal	Pin	Pin	Signal	
V_{SS} (0 V)	1	40	$\overline{Reset}$	
$\overline{RDY}$	2	39	Φ2	
Φ1	3	38	SO	(Set Overflow)
$\overline{IRQ}$	4	37	Φ0	(Takteingang)
n.c.	5	36	R/$\overline{W}$	
$\overline{NMI}$	6	35	$\overline{HALT}$	
n.c.	7	34	n.c.	
V_{CC} (+5 V)	8	33	D0	
A0	9	32	D1	
A1	10	31	D2	
A2	11	30	D3	
A3	12	29	D4	
A4	13	28	D5	
A5	14	27	D6	
A6	15	26	D7	
A7	16	25	A15	
A8	17	24	A14	
A9	18	23	A13	
A10	19	22	A12	
A11	20	21	V_{SS} (0 V)	

Die CPU 6502C wurde nur bei den neuen Atari-Geräten verwendet, also nur beim 600XL und beim 800XL.

Die Anschlußbelegung des ANTICs



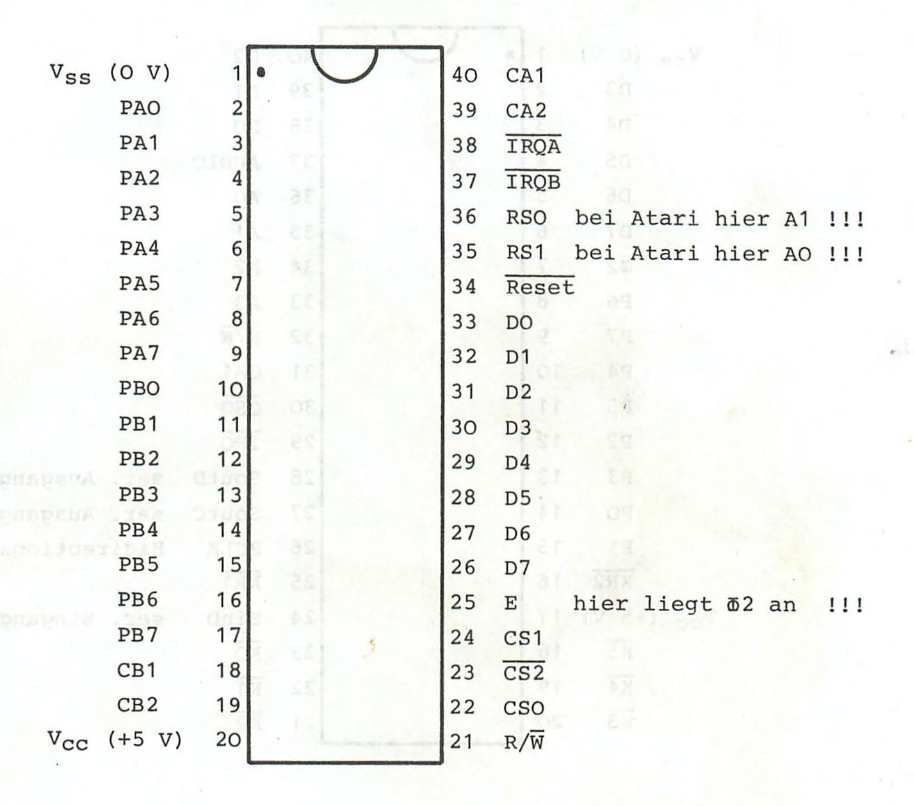
Die Anschlußbelegung des GTIA

Funktion	Signal	Pin	Pin	Signal	Funktion
	A1	1	40	A2	
	AO	2	39	A3	
V_{SS}	(O V)	3	38	A4	
	D3	4	37	D4	
	D2	5	36	D5	
	D1	6	35	D6	
	DO	7	34	D7	
Trigger O	TO	8	33	$R/\overline{W}$	
Trigger 1	T1	9	32	$\overline{CS}$	
Trigger 2	T2	10	31	Lum3	
Trigger 3	T3	11	30	Φ2	
'OPTION'	SO	12	29	FΦO	
'START'	S1	13	28	OSC	Oszillator-Eingang
'SELECT'	S2	14	27	V_{CC}	(+5 V)
Keyboard Click	S3	15	26	$\overline{HALT}$	
PAL-Verzögerung	PAL	16	25	SYNC	(Synchronisation)
Farbverzögerung	CADJ	17	24	Lum2	
	ANO	18	23	Lum1	
	AN1	19	22	LumO	
	AN2	20	21	COL	(Farbe)

Die Anschlußbelegung des POKEY

Signal	Pin	Pin	Signal	Beschreibung
V_{SS} (0 V)	1	40	D2	
D3	2	39	D1	
D4	3	38	D0	
D5	4	37	AUDIO	
D6	5	36	A0	
D7	6	35	A1	
$\Phi 2$	7	34	A2	
P6	8	33	A3	
P7	9	32	$R/\overline{W}$	
P4	10	31	CS1	
P5	11	30	$\overline{CS0}$	
P2	12	29	$\overline{IRQ}$	
P3	13	28	SoutD	ser. Ausgang Daten
P0	14	27	SoutC	ser. Ausgang Clock
P1	15	26	BCLK	Bidirectional Clock
$\overline{KR2}$	16	25	$\overline{KR1}$	
V_{CC} (+5 V)	17	24	SinD	ser. Eingang Daten
$\overline{K5}$	18	23	$\overline{K0}$	
$\overline{K4}$	19	22	$\overline{K1}$	
$\overline{K3}$	20	21	$\overline{K2}$	

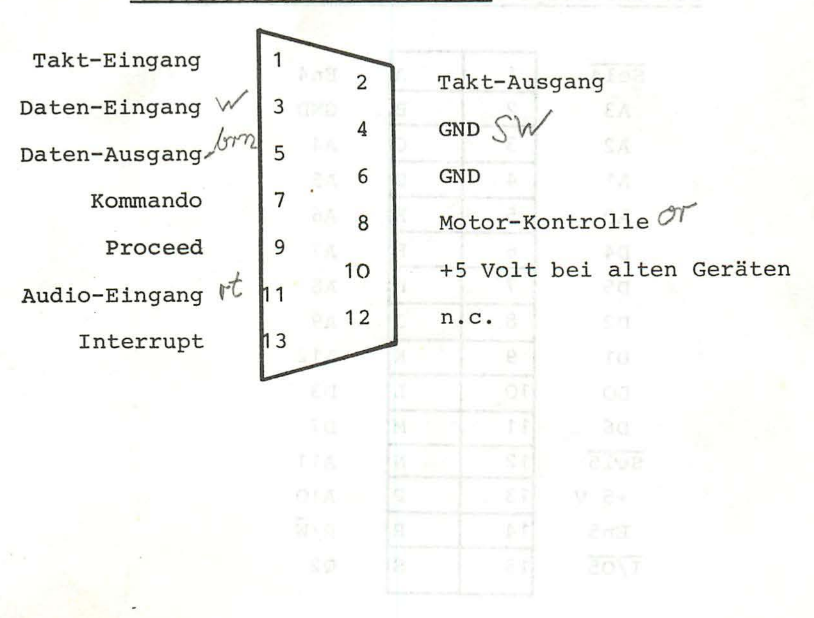
Die Anschlußbelegung der PIA



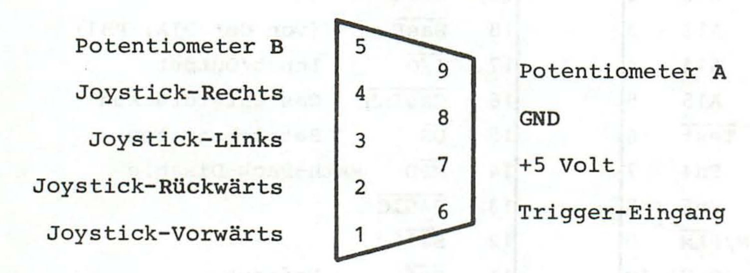
Die Anschlußbelegung der MMU

Signal	Pin	Pin	Signal	Beschreibung
A11	1	20	V_{CC} (+5 V)	
A12	2	19	$\overline{\text{Sel4}}$	
A13	3	18	$\overline{\text{BasE}}$	(von der PIA, PB1)
A14	4	17	$\overline{\text{I/O}}$	Input/Output
A15	5	16	$\overline{\text{CasInh}}$	Cas unterdrücken
$\overline{\text{Test}}$	6	15	$\overline{\text{OS}}$	Betriebssystem
En4	7	14	$\overline{\text{MPD}}$	Math-Pack-Disable
En5	8	13	$\overline{\text{BASIC}}$	
ROM/$\overline{\text{RAM}}$	9	12	$\overline{\text{Sel5}}$	
V_{SS} (0 V)	10	11	$\overline{\text{Ref}}$	Refresh

Die serielle Schnittstelle



Die Joystick-Ports



Die Cartridge-Buchse der Atari-Geräte

Einschub-Unterseite	Pin	Pin	Einschub-Oberseite
$\overline{\text{Sel4}}$	1	A	En4
A3	2	B	GND
A2	3	C	A4
A1	4	D	A5
A0	5	E	A6
D4	6	F	A7
D5	7	H	A8
D2	8	J	A9
D1	9	K	A12
D0	10	L	D3
D6	11	M	D7
$\overline{\text{Sel5}}$	12	N	A11
+5 V	13	P	A10
En5	14	R	R/$\overline{\text{W}}$
$\overline{\text{I/O5}}$	15	S	Φ2

Einschub-Unterseite

Einschub-Oberseite (Seite, die zur Vorderseite des Atari zeigt)

Der BUS-Stecker

(Oberseite)			(Unterseite)	
GND	1	2	$\overline{\text{ExtSel}}$	
A0	3	4	A1	
A2	5	6	A3	
A4	7	8	A5	
A6	9	10	GND	
A7	11	12	A8	
A9	13	14	A10	
A11	15	16	A12	
A13	17	18	A14	
GND	19	20	A15	
D0	21	22	D1	
D2	23	24	D3	
D4	25	26	D5	
D6	27	28	D7	
GND	29	30	GND	
Φ2	31	32	GND	
	33	34	$\overline{\text{Reset}}$	
$\overline{\text{IRQ}}$	35	36	$\overline{\text{RDY}}$	
	37	38	$\overline{\text{CasInh}}$	(Ausgang)
	39	40	$\overline{\text{Ref}}$	
$\overline{\text{CAS}}$	41	42	GND	
Math-Pack-Disable $\overline{\text{MPD}}$	43	44	$\overline{\text{RAS}}$	
GND	45	46	LR/$\overline{\text{W}}$	(gepuffertes R/$\overline{\text{W}}$)
+5 V	47	48	+5 V	
Audio In	49	50	GND	

D E R   A N T I C

Der ANTIC ist kein Standard-IC, sondern ein Spezialbaustein. "ANTIC" steht dabei wohl für "Alphanumeric Television Interface Controller". Beim ANTIC handelt es sich um einen kundenspezifisch hergestellten Mikroprozessor, der als Subprozessor für den Bildaufbau zuständig ist. Der ANTIC kann der CPU Arbeitszyklen "stehlen" (man spricht hier wirklich von "Cycle-Stealing"), das heißt, daß er die CPU immer dann anhält, wenn er Daten aus dem Arbeitsspeicher benötigt. Der ANTIC liest die Daten also im direkten Speicherzugriff, im DMA (Direct Memory Access). Der ANTIC kann die gesamten 64 KByte Arbeitsspeicher adressieren und muß nicht erst, wie andere Grafikbausteine in anderen Mikrocomputersystemen, durch Register auf bestimmte Speicherbereiche gelegt werden.

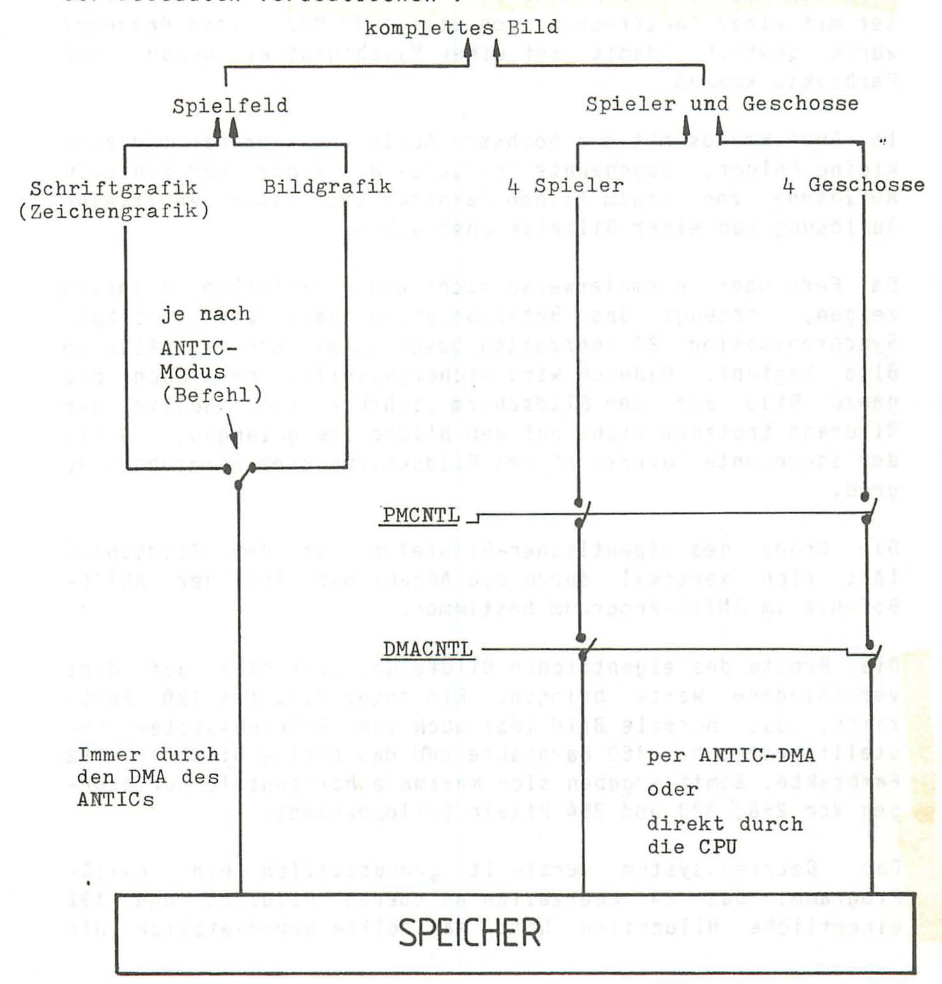
Der ANTIC besitzt einen eigenen Befehlssatz, mit dem man ihn dazu bringen kann, einzelne oder auch gleich mehrere Bildzeilen auf dem Bildschirm darzustellen. Durch den ANTIC ist es möglich, verschiedene Grafikmodi beliebig auf dem Bildschirm zu kombinieren. Man kann also Schrift- und Grafikmodi gleichzeitig verwenden. Das ist bislang mit keinem anderen Mikrocomputer dieser Preisklasse mit der gleichen Flexibilität möglich.

Der ANTIC ist aber nicht alleine für die Bildausgabe zuständig. Er erstellt nicht selbst das fertige Bildsignal mit den Farbinformationen, sondern überträgt die Informationen zur Bilderstellung, die er aufgrund von Befehlen aus seinem Programm (dem ANTIC-Programm) aus dem Arbeitsspeicher geholt hat, zu einem weiteren Baustein, dem GTIA. Der GTIA fügt zu den Bildinformationen, die der ANTIC ihm überträgt, die Farbinformationen hinzu.

Außerdem ist der GTIA für die sogenannte Player-Missile-Grafik, das heißt für die Darstellung beweglicher Objekte auf dem Bildschirm, zuständig. Der ANTIC kann den GTIA bei der Erstellung der Player-Missile-Grafik unterstützen, indem er dem GTIA in jeder Bildzeile die Daten zur Erstellung der Player-Missile-Grafik aus dem Arbeitsspeicher holt und in GTIA-Register überträgt.

Für viele der Register des ANTICs sind sogenannte Schattenregister vorhanden. Während des Vertikal-Leertaktes des Bildes werden die Register- und Schattenregisterinhalte aufeinander abgestimmt. Weitere Ausführungen dazu befinden sich in den Kapiteln über den GTIA und das Betriebssystem.

Das folgende Schaubild soll die Herkunft und Verarbeitung der Bilddaten verdeutlichen:



Der ANTIC erstellt 50 Halbbilder pro Sekunde. Die Halbbilder sind aber nicht wie beim normalen Fernsehbild gegeneinander verschoben. Auf diese Art erhält man ein ruhiger stehendes Bild. Man spricht hier auch vom "non interlaced Scan". Jedes dieser Halbbilder besteht aus 310 Bildzeilen. Jede Bildzeile besteht wiederum aus 228 sogenannten Farbtakten (engl. "Color-Clocks"). Ein solcher Farbtakt hat ungefähr die Breite von zwei Bildzeilen-Höhen. In einer Sekunde gibt es also ca. 3,5 Mio. Farbtakte (50 * 310 * 228). Die CPU arbeitet mit einer Taktfrequenz von etwa 1,77 MHz. Diese Frequenz wurde gewählt, damit auf einen Maschinentakt genau zwei Farbtakte kommen.

Im Grafikmodus mit der höchsten Auflösung kann man einzelne kleine Felder, sogenannte "PIXEL", mit einer horizontalen Auflösung von einem halben Farbtakt und einer vertikalen Auflösung von einer Bilzeile ansprechen.

Da Fernseher normalerweise nicht den wirklichen Bildrand zeigen, erzeugt das Betriebssystem nach der Vertikal-Synchronisation 24 Leerzeilen bevor es mit dem eigentlichen Bild beginnt. Dadurch wird sichergestellt, daß auch das ganze Bild auf dem Bildschirm sichtbar ist. Sollte der Bildrand trotzdem nicht auf den Bildschirm gelangen, so ist der sogenannte "Overscan" des Bildschirms oder Fernsehers zu groß.

Die Größe des eigentlichen Bildfeldes auf dem Bildschirm läßt sich vertikal durch die Anzahl und Art der ANTIC-Befehle im ANTIC-Programm bestimmen.

Die Breite des eigentlichen Bildfeldes läßt sich auf drei verschiedene Werte bringen. Ein enges Bild hat 128 Farbtakte, das normale Bild (das auch vom Betriebssystem erstellt wird) hat 160 Farbtakte und das breite Bild hat 192 Farbtakte. Somit ergeben sich maximale horizontale Auflösungen von 256, 320 und 384 Pixeln (Bildpunkten).

Das Betriebssystem erstellt grundsätzlich ein ANTIC-Programm, das 24 Leerzeilen am oberen Bildrand und 192 eigentliche Bildzeilen hat. Man sollte grundsätzlich nie

mehr Zeilen als das Betriebssystem abbilden. Wenn man versucht, mehr als 24 Leer- und 192 Bildzeilen auf den Bildschirm zu bringen, ist es möglich, daß das Bild anfängt zu rollen.

```
*********************************************
*                                           *
*                                           *
*   DIE KONTROLL- UND LIGHTPENREGISTER      *
*   ----------------------------------      *
*                                           *
*********************************************
```

Das Register (bzw. das Schattenregister)

DMACNTL 54272 $d400
DMACNTL§ 559 $22f

kontrolliert den direkten Speicherzugriff des ANTIC. Dabei haben die einzelnen Bits des Registers folgende Funktionen:

Bit 0-1 :	Größe des Bildschirms
Bit 2 :	Wenn dieses Bit auf "1" steht, ist der DMA für die Geschosse eingeschaltet.
Bit 3 :	Wenn dieses Bit auf "1" steht, ist der DMA für die Spieler eingeschaltet.
Bit 4 :	Wenn dieses Bit auf "1" steht, ist einzeilige Auflösung für Spieler und Geschosse gewählt, ist das Bit dagegen "0", werden Spieler und Geschosse zweizeilig aufgelöst dargestellt.

Bit 5 : Wenn dieses Bit auf "1" steht, ist der DMA für das Lesen des ANTIC-Programms eingeschaltet. Steht das Bit auf "0", erscheint kein Bild auf dem Monitor.

Bit 6-7 : unbenutzt

Die Bits 0 und 1 des Registers wählen die Breite des Bildfeldes auf folgende Art:

Bit1	Bit0	Funktion
0	0	Es werden keine DMA-Zyklen für das Bild ausgeführt, es gibt also kein Bild (Spielfeld).
0	1	Enges Spielfeld Das Bild hat jetzt 128 Farbtakte.
1	0	Normales Spielfeld Das Bild hat jetzt 160 Farbtakte. Diese Einstellung benutzt das Betriebssystem grundsätzlich.
1	1	Breites Spielfeld Das Bild hat jetzt 192 Farbtakte.

Der Standardwert (der sogenannte "default value") dieses Registers ist $22 bzw. 34.

Oft ist es nötig, die "normale" Tätigkeit der CPU zu unterbrechen, da bestimmte Aufgaben zum Beispiel nur während der Vertikal-Synchronisation erledigt werden können oder sollen, oder es müssen bestimmte Anfragen von Peripheriegeräten sofort beantwortet werden. Dafür gibt es bei der CPU 6502, die auch im Atari verwendet wurde, grundsätzlich zwei verschiedene Interrupts (Unterbrechungen). Der maskierbare Interrupt (IRQ : Interrupt ReQuest, Unterbrechung auf An-

frage) wird vom POKEY und gegebenenfalls von der PIA benutzt. Der nicht von der CPU intern maskierbare (abschaltbare) Interrupt (NMI : Non Maskable Interrupt) wird vom ANTIC verwaltet. Bei den alten 400/800er Modellen löste auch die "RESET"-Taste über den ANTIC einen NMI aus, bei den neuen Modellen ist diese Methode unbrauchbar geworden, weil zum Beispiel nach dem Einstecken eines (anderen) ROM-Moduls unbedingt ein Hardware-Reset erfolgen muß, wenn dabei das System abgestürzt ist. Bei den alten Geräten wurde beim Modulwechsel automatisch der Computer abgeschaltet.

Im ANTIC gibt es das Register

NMIEN 54286 $d40e

mit dem es möglich ist, die sogenannten ANTIC-Programm-Unterbrechungen und den Vertikalsynchron-Interrupt (in dem das Betriebssystem zum Beispiel die Schattenregister erzeugt) bereits im ANTIC zu unterbinden. Eine Unterbindung (Maskierung) des NMI bei (hier nicht vorgesehenem) Signal an der "RNMI"-Leitung ist mit NMIEN nicht möglich. Die Bits von NMIEN haben folgende Funktionen:

Bit 0-5 :	nicht benutzt
Bit 6 :	Wenn dieses Bit gesetzt ist, ist der Vertikalsynchron-Interrupt eingeschaltet
Bit 7 :	Wenn dieses Bit gesetzt ist, sind ANTIC-Programm-Unterbrechungen möglich.

Vom Betriebssystem wird der Wert $40 bzw. 64 in NMIEN eingetragen. Wenn der NMI-Eingang der CPU auf "0" geht, unterbricht sie ihre normale Tätigkeit und bearbeitet eine Interrupt-Routine, deren Anfangsadresse an einer bestimmten Adresse ($fffa und $fffb) im System gespeichert ist . Diese Routine muß am Anfang feststellen, was die Unterbrechung ausgelöst hat. Dies kann sie im Register

NMIST 54287 $d40f

abfragen. Für jede Interruptquelle steht ein Bit zur Verfügung. Daß eine bestimmte Unterbrechungsbedingung aufgetreten ist, zeigt der ANTIC durch Setzen des entsprechenden Bits. Die Bits in NMIST sind wie folgt zugeordnet:

Bit 0-4 : unbenutzt
Bit 5 : "RNMI"-Eingang
Bit 6 : Vertikalsynchron-Interrupt
Bit 7 : ANTIC-Programm-Unterbrechung

Zum Rücksetzen eines gesetzten Bits in diesem Register, muß man lediglich irgendeinen Wert in das Register

NMIRES 54287 $d40f

schreiben. Dieses Register hat die gleiche Adresse wie NMIST. Wenn auf diese Adresse ein Lesezugriff erfolgt, wird NMIST angesprochen, bei einem Schreibzugriff dagegen NMIRES.

Mit dem Register

VCOUNT 54283 $d40b

ist es möglich festzustellen, welche Bildzeile der ANTIC gerade darstellt. Die Bits des Registers VCOUNT enthalten den Stand des internen Bildzeilenzählers. Das niedrigste Bit des Bildzeilenzählers ist aber nicht in VCOUNT enthalten. VCOUNT enthält also die Nummer der aktuellen Bildzeile geteilt durch zwei. Bei einem Bilddurchlauf zählt VCOUNT von 0 bis 155.

Unter Umständen ist es für besondere Effekte nötig den Prozessor parallel zu einer Zeile arbeiten zu lassen. Da der Aufbau einer Zeile jedoch zu schnell ist, um einfach abzufragen, wann die nächste Zeile beginnt, gibt es das Register

WAITHSYNC 54282 $d40a

Wenn irgendein Wert in dieses Register hineingeschrieben wird, wird die CPU bis zum Anfang der nächsten Zeile einfach gestoppt. Dabei wird die "READY"-Leitung der CPU auf "0" geschaltet. Sobald das nächste Horizotalsynchronisationssignal kommt, also die Zeile beendet ist, arbeitet die CPU wieder weiter.

An den Atari-Computer läßt sich an die Joystick-Anschlüsse ein sogenannter "Lightpen" anschließen. Wenn man einen derartigen Lightpen auf den Bildschirm hält, wird seine Position in zwei Registern und Schattenregistern gespeichert:

LPENH LPENH§	54284 564	$d40c $234	Hier wird die Horizontalposition des Lightpens gespeichert. Das Register enthält die Nummer des Farbtaktes, bei dem sich der Lightpen befindet.
LPENV LPENV§	54285 565	$d40d $235	Hier wird die Vertikalposition des Lightpens gespeichert. Das Register enthält die Nummer der Bildzeile, in der sich der Lightpen befindet geteilt durch zwei. (vergl. VCOUNT)

Es gibt allerdings häufig Probleme beim Einsatz eines Lightpens. Ein Lightpen ist eigentlich nichts weiter als ein Fototransistor. Das Prinzip des Lightpens ist es, ein Signal an den ANTIC zu geben, sobald der das Bild erzeugende Strahl d Position des Lightpens passiert. Dazu muß der Fototransistor oder das jeweilige Aufnahmeelement jedoch sehr schnell reagieren. Derartig schnell reagierende Elemente sind sehr teuer. Unter Umständen erhält man auch akzeptable Ergebnisse mit billigen Fototransistoren, meist gibt es mit ihnen aber Probleme bei der Aufnahme der Horizontalposition.

Für viele Zwecke ist es ausreichend nur die Vertikalposition des Lightpens festzustellen, zum Beispiel wenn man Punkte eines Auswahlmenüs mit dem Lightpen wählen möchte. Dann kann auch ein in der Vertikalposition nicht richtig funktionierender Lightpen befriedigende Ergebnisse liefern.

```
*********************************************
*                                           *
*                                           *
*  DER DMA FÜR DIE PLAYER-MISSILE-GRAFIK    *
*  ----------------------------------       *
*                                           *
*********************************************
```

Die Player-Missile-Grafik wird vom GTIA erzeugt. Der GTIA unterstützt dabei jedoch immer nur eine Bildzeile. Es müssen daher für jede Bildzeile jeweils neue Daten in die sogenannten Grafikregister des GTIA geschrieben werden. Dies kann auch per Assemblerprogramm geschehen, der Aufwand dafür ist aber sehr hoch. Einfacher ist es, den DMA des ANTICs für die Player-Missile-Grafik zu verwenden.

Man muß lediglich ein Speicherfeld mit den Daten der Spieler und Geschosse im Speicher aufbauen. Für jede Bildzeile (oder für jede zweite Bildzeile, wenn zweizeilige Auflösung gewählt ist) holt der ANTIC Daten aus dem Speicher und überträgt sie in die GTIA-Grafikregister. Die Horizontalposition der Objekte wählt man dann im GTIA, die Vertikalposition wird durch die Lage des Objekts im Player-Missile-Speicherfeld bestimmt.

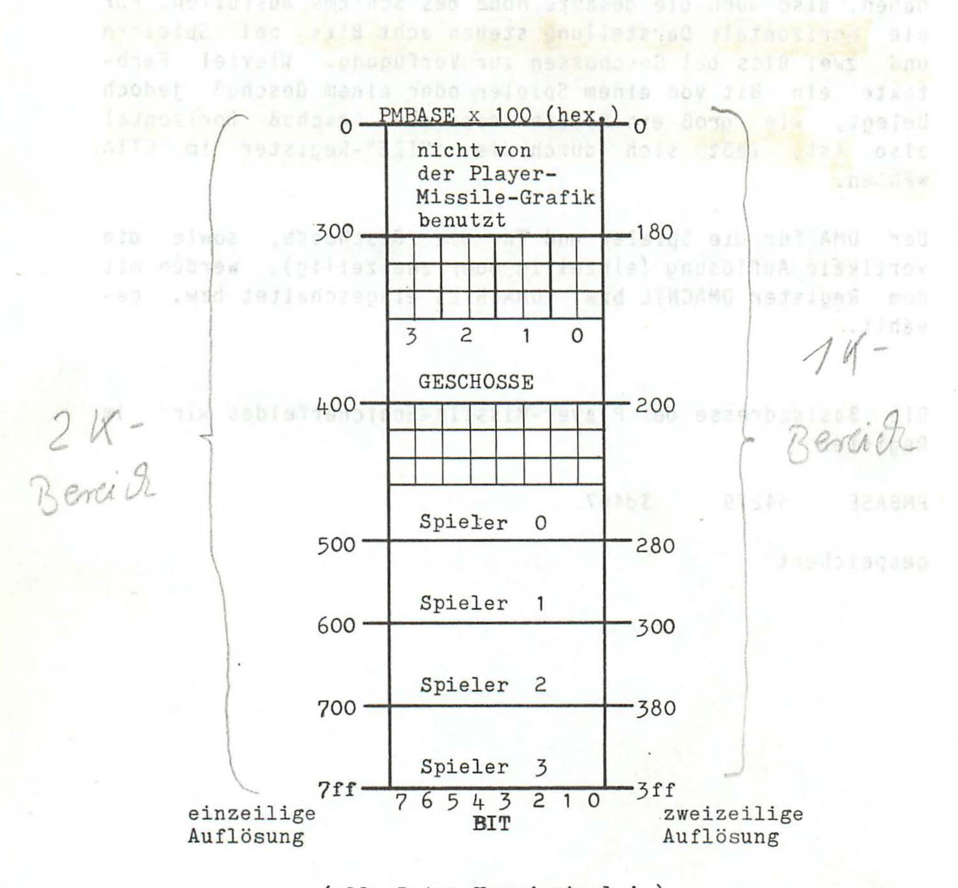
Auf diese Art können Spieler beliebige vertikale Ausdehnung haben, also auch die gesamte Höhe des Schirms ausfüllen. Für die horizontale Darstellung stehen acht Bits bei Spielern und zwei Bits bei Geschossen zur Verfügung. Wieviel Farbtakte ein Bit von einem Spieler oder einem Geschoß jedoch belegt, wie groß ein Spieler oder ein Geschoß horizontal also ist, läßt sich durch die "SIZE"-Register im GTIA wählen.

Der DMA für die Spieler und für die Geschosse, sowie die vertikale Auflösung (einzeilig oder zweizeilig), werden mit dem Register DMACNTL bzw. DMACNTL§ eingeschaltet bzw. gewählt.

Die Basisadresse des Player-Missile-Speicherfeldes wird im Register

PMBASE 54279 $d407

gespeichert.



(alle Daten Hexadezimal !)

Die eigentliche Adresse, aus der sich der ANTIC die Daten für die Player-Missile-Grafik holt, setzt sich bei einzeiliger Auflösung folgendermaßen zusammen:

Bit 0-7	:	Bildzeilenzähler für Spieler und Geschosse
Bit 8-10	:	Diese Bits wählen, für welchen Spieler/Geschoß der DMA erfolgt.
Bit 11	:	PMBASE, Bit 3
Bit 12	:	PMBASE, Bit 4
Bit 13	:	PMBASE, Bit 5
Bit 14	:	PMBASE, Bit 6
Bit 15	:	PMBASE, Bit 7

PMBASE wählt also die Nummer des 256-Byte-Blocks, an dem das Player-Missile-Speicherfeld beginnt. Dieser Block muß aber am Beginn eines 2 KByte Speicherblockes liegen. Man kann auch sagen, daß mit PMBASE einer von 32 möglichen 2 KByte Speicherblöcken ausgewählt wird (5 Bits von PMBASE werden verwendet). Die Bits 8-10 "wählen" wie erwähnt, für welchen Spieler/Geschoss der DMA erfolgt. Dies geschieht auf folgende Weise:

Bit10	Bit9	Bit8	Beschreibung
0	0	0	Diese 3 Kombinationen treten nicht auf,
0	0	1	daher sind bei einzeiliger Auflösung
0	1	0	auch die ersten 3 * 256 Bytes unbenutzt. Prinzipiell gibt es 8 Kombinationen der Bits 8-10, da aber nur 4 Felder für Spieler und ein Feld für die Geschosse benötigt werden, bleiben 3 Kombinationen unbenutzt.
0	1	1	Bei dieser Kombination wird aus dem Speicherfeld für die Geschosse im DMA gelesen.
1	0	0	Bei dieser Kombination wird aus dem Speicherfeld für Spieler 0 im DMA

			gelesen.
1	0	1	Bei dieser Kombination wird aus dem Speicherfeld für Spieler 1 im DMA gelesen.
1	1	0	Bei dieser Kombination wird aus dem Speicherfeld für Spieler 2 im DMA gelesen.
1	1	1	Bei dieser Kombination wird aus dem Speicherfeld für Spieler 3 im DMA gelesen.

Bei zweizeiliger Auflösung der Spieler und Geschosse, setzt sich die eigentliche Adresse für den DMA (das heißt die Adresse, aus der die Daten, die in die Grafikregister geschrieben werden, gelesen werden) folgendermaßen zusammen:

Bit 0-6 :	Bildzeilenzähler für Spieler und Geschosse
Bit 7-9 :	Diese Bits wählen, für welchen Spieler/Geschoß der DMA erfolgt. Dabei gilt das für die Bits 8-10 bei einzeiliger Auflösung Gesagte. Jedoch muß beachtet werden, daß jeder Spieler und die Geschosse nur je 128 Bytes belegen. Es bleiben am Anfang des Player-Missile-Speicherfeldes also auch nur 3 * 128 Byte unbenutzt (im Gegensatz zu den 3 * 256 Byte bei einzeiliger Auflösung).
Bit 10 :	PMBASE, Bit 2
Bit 11 :	PMBASE, Bit 3
Bit 12 :	PMBASE, Bit 4
Bit 13 :	PMBASE, Bit 5
Bit 14 :	PMBASE, Bit 6
Bit 15 :	PMBASE, Bit 7

PMBASE wählt also die Nummer des 256-Byte-Blocks, an dem das Player-Missile-Speicherfeld beginnt. Dieser Block muß aber am Beginn eines 1 KByte Speicherblockes liegen. Man kann auch sagen, daß mit PMBASE einer von 64 möglichen 1 KByte Speicherblöcken ausgewählt wird.

```
**********************************************************
*                                                        *
*                                                        *
*   DER BEFEHLSATZ DES ANTIC UND DAS ANTIC-PROGRAMM     *
*   ---------------------------------------------        *
*                                                        *
**********************************************************
```

Wie schon erwähnt, handelt es sich beim ANTIC um einen Mikroprozessor. Bei seinen Befehlen handelt es sich um Kommandos, die die Darstellung von 1 bis 16 Bildzeilen zur Folge haben. Außerdem gibt es für jeden Befehl eine bestimmte Form, die veranlaßt, daß sich der ANTIC den Anfang des Bildspeichers "merkt", daß der sogenannte Bildspeicherzähler geladen wird. Wie bei "normalen" Mikroprozessoren gibt es auch beim ANTIC Sprungbefehle, die bewirken, das er das ANTIC-Programm von einer anderen Speicherstelle aus weiterliest.

Natürlich muß man dem ANTIC mitteilen, von welcher Speicherstelle aus er sein Programm abarbeiten soll. Diese Adresse teilt man ihm in zwei Registern bzw. zwei Schattenregistern (einem sogenannten Pointer, also Zeiger auf das ANTIC-Programm) mit:

DLPTRL	54274	$d402	unteres Byte der Adresse
DLPTRL$	560	$230	(Schattenregister)

DLPTRH	54275	$d403	obere Hälfte der Adresse
DLPTRH§	561	$231	(Schattenregister)

Die Akürzung "DLPTR" kommt von "Display-List-Pointer", dies heißt soviel wie "Zeiger auf das ANTIC-Programm". Der ANTIC verwendet den Wert der Register intern als Wert seines Programm-Zählers. Dieser ANTIC-Programm-Zähler zeigt jeweils auf den abzuarbeitenden ANTIC-Befehl. Da die obersten 6 Bits des ANTIC-Programm-Zählers nur Register sind und nicht hochgezählt werden können, kann das ANTIC-Programm eine 1 KByte-Grenze nur durch einen ANTIC-Programm-Sprung überwinden.

Durch den Wechsel der Inhalte der Register DLPTRL und DLPTRH kann man das gesamte Bild wechseln. Man muß vorher lediglich ein neues ANTIC-Programm und einen Bildspeicher irgendwo im System aufbauen.

Man sollte diese Register nur während der Vertikalsynchronisation ändern, ansonsten kann das Bild kurzzeitig abrollen. Die Schattenregisterinhalte werden immer während der Vertikalsynchronisation in die Register geschrieben, daher muß hier nicht darauf geachtet werden, daß man selbst nur während der Vertikalsynchronisation schreibt. Damit der ANTIC überhaupt das ANTIC-Programm lesen kann, muß Bit 5 von Register DMACNTL gesetzt sein.

Alle ANTIC-Befehle kann man grundsätzlich in 3 Gruppen einteilen:

1) Befehle, um Leerzeilen zu erzeugen
2) Sprungbefehle
3) Befehle, die Bildzeilen erzeugen

Jeder ANTIC-Befehl wird per DMA in das Befehlsregister geladen. Bei Sprungbefehlen werden vom ANTIC zwei weitere Bytes per DMA gelesen. Diese beiden Bytes werden in den ANTIC-Programm-Zähler geladen. Wie bei CPU-Befehlen wird auch hier zuerst das niederwertige Byte und dann das höherwertige Byte gelesen.

Bei Befehlen, die veranlassen, daß der Bildspeicherzähler geladen wird, holt der ANTIC ebenfalls zwei weitere Bytes per DMA aus dem Speicher und transportiert sie in das Bildspeicherzählerregister (auch hier wird das niederwertige Byte zuerst gelesen). Bei "normalen" Anzeigekommandos holt der ANTIC das nächste Byte der Bilddaten direkt aus der Speicheradresse, die der folgt, aus der das letzte Byte der vorherigen Zeile gelesen wurde. Der Bildspeicherzähler wird dabei nicht neu geladen. Dadurch werden die Bilddaten der jeweils nächsten Zeile direkt aus den Speicherstellen, die der vorhergehenden Zeile folgen, gelesen. Die Bilddaten der einzelnen Zeilen werden so ohne Lücke aneinandergereiht. Es ergibt sich dadurch ein Bildspeicherblock, wie er auch bei fast allen anderen Mikrocomputern vorhanden ist.

Der Bildspeicherzähler (der Pointer auf die Bilddaten) besteht aus 4 Register- und 12 Zählerbits. Daher kann ein zusammenhängender Bildspeicherbereich nie länger als 4 KByte sein. Jeweils beim Übergang zum nächsten 4 KByte-Block muß der Bildspeicherzähler durch ein geeignetes ANTIC-Kommando neu geladen werden. Besonders sollte man auch darauf achten, das der 4 KByte-Block nicht während der Darstellung einer Zeile überschritten wird. Dies würde nämlich dazu führen, daß plötzlich, innerhalb der laufenden Zeile, die Daten aus Speicherstellen vom Anfang des durch die obersten 4 Bits des Bildspeicherzählers bestimmten 4 KByte-Blocks, gelesen würden.

Für jeden Befehl gibt es eine Version, die veranlaßt, daß bei der Bearbeitung der letzten, durch diesen Befehl erzeugten Bildzeile, die normale Tätigkeit der CPU unterbrochen wird und stattdessen die Routine, deren Startadresse bei DLIVKT (niederwertiges Byte in 512/$200, höherwertiges Byte in 513/$201) steht, bearbeitet wird. Die Vorraussetzung für eine derartige ANTIC-Programm-Unterbrechung ist, daß Bit 7 von NMIEN gesetzt ist. Eine ANTIC-Programm-Unterbrechung ist nicht eine Unterbrechung des ANTIC-Programms, sondern eine Unterbrechung (Interrupt) der CPU, die durch das ANTIC-

Programm ausgelöst wird.

Mit ANTIC-Programm-Unterbrechungen lassen sich zum Beispiel während eines Bildes in einer bestimmten Zeile die Farben wechseln. Mit einer ANTIC-Programm-Unterbrechung ist es auch möglich, mehr als vier Spieler und Geschosse darzustellen, indem man PMBASE auf ein anderes Player-Missile-Speicherfeld legt und die Farben und Bildschirmpositionen, soweit nötig, wechselt. ANTIC-Programm-Unterbrechungen bieten vielfältige Möglichkeiten, da es mit ihnen auf sehr einfache Art möglich ist, einen Programmteil während der Darstellung einer bestimmten Bildschirmposition abzuarbeiten.

Befehle, um Leerzeilen zu erzeugen:

Der ANTIC besitzt Befehle, die auf dem Bildschirm Leerzeilen erzeugen. Wenn ein derartiger Befehl abgearbeitet wird, werden keine Bilddaten aus dem Speicher geholt. Statt dessen erscheint die Hintergrundfarbe auf dem Monitor. Die Leerzeilenbefehle sind grundsätzlich ein Byte lang. Mit einem Leerzeilenbefehl werden ein bis acht Leerzeilen erzeugt. Ein ANTIC-Befehl für Leerzeilen wird wie folgt aufgebaut:

Bit 0-3 :	Diese Bits müssen für einen Leerzeilenbefehl binär %0000 enthalten.
Bit 4-6 :	Der Binärwert dieser drei Bits bestimmt die Anzahl der Leerzeilen plus eins, die mit diesem Befehl erzeugt werden. Wenn diese Bits zum Beispiel binär %010 enthalten, werden 3 Leerzeilen erzeugt.
Bit 7 :	Wenn dieses Bit gesetzt ist, wird eine ANTIC-Programm-Unterbrechung bei der Darstellung der letzten, durch diesen Befehl erzeugten Bildzeile ausgelöst.

Sprungbefehle:

Mit diesen Befehlen wird der ANTIC-Programm-Zähler neu geladen. Wenn ein derartiger Befehl in einem Bildblock auftritt, verursacht er eine Leerzeile. Daher sollten Sprungbefehle nur am Ende eines Bildes ausgelöst werden. Die beiden Bytes, die im ANTIC-Programm einem Sprungbefehl folgen, enthalten die neue Adresse (niederwertiges Byte zuerst). Ein ANTIC-Sprungbefehl wird wie folgt aufgebaut:

Bit 0-3 :	Diese Bits müssen bei einem Sprungbefehl binär %0001 enthalten.
Bit 4 :	Wenn dieses Bit "0" ist, wird ein einfacher Sprung ausgelöst, bei dem eine Leerzeile verursacht wird. Ist dieses Bit "1", erfolgt ebenfalls ein Sprung an die angegebene Adresse. Der ANTIC wartet mit der Ausführung des folgenden Befehls bis zum Ende der nächsten Vertikalsynchronisation, also bis zum nächsten Bild.
Bit 5-6 :	Diese Bits sollten "0" sein.
Bit 7 :	Wenn dieses Bit "1" ist, wird eine ANTIC-Programm-Unterbrechung ausgelöst.

Befehle, die Bildzeilen erzeugen:

Diese Befehle enthalten die Nummer (hexadezimal) des ANTIC-Modus, mit dem die nächste Bildzeile, bzw. der nächste Bildblock dargestellt wird. Oft werden diese Befehle auch als "Display"-Befehle bezeichnet. Ein Displaybefehl des ANTIC ist folgendermaßen aufgebaut:

Bit 0-3 :	Diese Bits enthalten die (hexadezimale) Nummer/Bezeichnung des ANTIC-Modus.
Bit 4 :	Wenn dieses Bit gesetzt ist, ist der

Horizontalvorschub (das Horizontal-Scrolling) eingeschaltet.

Bit 5 : Wenn dieses Bit gesetzt ist, ist der Vertikalvorschub (das Vertikal-Scrolling) eingeschaltet.

Bit 6 : Wenn dieses Bit gesetzt ist, so wird vor der Darstellung der ersten, zu diesem Befehl gehörigen Zeile, der Bildspeicherzähler mit den beiden im ANTIC-Programm folgenden Bytes geladen.

Bit 7 : Wenn dieses Bit gesetzt ist, wird eine ANTIC-Programm-Unterbrechung bei der Darstellung der letzten durch diesen Befehl erzeugten Bildzeile ausgelöst.

Die folgenden Tabellen fassen die ANTIC-Befehle noch einmal zusammen (dabei sind die Daten hexadezimal angegeben):

BEFEHL	~~mit~~ ohne	~~ohne~~ mit
	ANTIC-Programm-Unterbrechung	
1 Leerzeile	00	80
2 Leerzeilen	10	90
3 Leerzeilen	20	a0
4 Leerzeilen	30	b0
5 Leerzeilen	40	c0
6 Leerzeilen	50	d0
7 Leerzeilen	60	e0
8 Leerzeilen	70	f0
Sprung	01	81
Sprung u. Warten	41	c1

HScrolling	--	XX	--	XX	--	XX	--	XX
VScrolling	--	--	XX	XX	--	--	XX	XX
lade Bildspeicherzähler	--	--	--	--	XX	XX	XX	XX

BEFEHL	OHNE ANTIC-PROGRAMM-UNTERBRECHUNG							
Modus "2"	02	12	22	32	42	52	62	72
Modus "3"	03	13	23	33	43	53	63	73
Modus "4"	04	14	24	34	44	54	64	74
Modus "5"	05	15	25	35	45	55	65	75
Modus "6"	06	16	26	36	46	56	66	76
Modus "7"	07	17	27	37	47	57	67	77
Modus "8"	08	18	28	38	48	58	68	78
Modus "9"	09	19	29	39	49	59	69	79
Modus "A"	0a	1a	2a	3a	4a	5a	6a	7a
Modus "B"	0b	1b	2b	3b	4b	5b	6b	7b
Modus "C"	0c	1c	2c	3c	4c	5c	6c	7c
Modus "D"	0d	1d	2d	3d	4d	5d	6d	7d
Modus "E"	0e	1e	2e	3e	4e	5e	6e	7e
Modus "F"	0f	1f	2f	3f	4f	5f	6f	7f

BEFEHL	MIT ANTIC-PROGRAMM-UNTERBRECHUNG							
Modus "2"	82	92	a2	b2	c2	d2	e2	f2
Modus "3"	83	93	a3	b3	c3	d3	e3	f3
Modus "4"	84	94	a4	b4	c4	d4	e4	f4
Modus "5"	85	95	a5	b5	c5	d5	e5	f5
Modus "6"	86	96	a6	b6	c6	d6	e6	f6
Modus "7"	87	97	a7	b7	c7	d7	e7	f7
Modus "8"	88	98	a8	b8	c8	d8	e8	f8
Modus "9"	89	99	a9	b9	c9	d9	e9	f9
Modus "A"	8a	9a	aa	ba	ca	da	ea	fa
Modus "B"	8b	9b	ab	bb	cb	db	eb	fb
Modus "C"	8c	9c	ac	bc	cc	dc	ec	fc
Modus "D"	8d	9d	ad	bd	cd	dd	ed	fd
Modus "E"	8e	9e	ae	be	ce	de	ee	fe
Modus "F"	8f	9f	af	bf	cf	df	ef	ff

BEISPIEL FÜR EIN ANTIC-PROGRAMM:

Das ANTIC Programm und der Bildspeicher im BASIC-Modus "0" ("GRAPHICS 0"):

```
Adresse  Daten
(hex.)   (hex.)

7c20     70        24 Leerzeilen am oberen Bildrand
         70        (wegen "Overscan" des Bildschirms)
         70
         42        Der Bildspeicherzähler wird mit
         40        $7c40 geladen
         7c        (Darstellung mit ANTIC-Modus "2")
         02        --I
         02          I
         02          I
         .           I
         .           I--  23 weitere Zeilen
         .           I    im ANTIC-Modus "2"
         .           I
         02          I
         02        --I
         41        Springe nach $7c20 und warte auf
         20        das Ende des Vertikal-Leertaktes
         7c        (ANTIC-Programm-Zähler neu laden)
7c40              1. Bildzeile  --I
7c68              2. Bildzeile    I
7c90              3. Bildzeile    I
.                                 I--  je 40 Zeichen
.                                 I    Bildspeicher
.                                 I
7f60             23. Bildzeile    I
7f88             24. Bildzeile  --I
```

Der Bildspeicher ist insgesamt 960 ($3c0) Bytes lang und endet bei $7faf (dezimal : 32687)

```
***************************
*                         *
*                         *
*  DIE BILDGRAFIK-MODI    *
*  -------------------    *
*                         *
***************************
```

Der ANTIC verfügt über acht verschiedene Bildgrafik-Modi. Bei einem Bildgrafik-Modus kann man jedes einzelne Pixel auf dem Bildschirm einzeln ansprechen. Nicht alle sind durch das Betriebssystem im BASIC verfügbar. Die einzelnen Modi unterscheiden sich im Grad der Auflösung und in der Anzahl der für jeden Pixel verfügbaren Farben, somit also auch im Speicherbedarf.

Unter einem Pixel versteht man ein rechteckiges Feld auf dem Bildschirm, das ein bis acht Bildzeilen hoch und einen halben bis 4 Farbtakte breit sein kann. Bei einem ANTIC-Programm mit 192 Bildzeilen und 160 Farbtakten pro Zeile ergeben sich somit Auflösungen von 24 * 40 bis 192 * 320 Pixeln. Bei 192 Farbtakten pro Zeile ergibt sich sogar eine maximale horizontale Auflösung von 384 Pixeln.

ANTIC-Modi mit geringer Auflösung haben den Vorteil, daß sie wenig Speicherraum verbrauchen, sich somit auch schnell komplett ändern lassen. Außerdem benötigen sie nicht so viele DMA-Zyklen, verlangsamen die CPU also auch weniger, als ANTIC-Modi mit hoher Auflösung. Die Adresse des Bildspeichers wird (wie im Kapitel über das ANTIC-Programm beschrieben) im ANTIC-Programm festgelegt.

Die Daten werden, wenn sie zur Bilddarstellung benötigt werden, per DMA vom ANTIC aus dem Arbeitsspeicher geholt. Außer auf den Bildschirm, transportiert der ANTIC die Bilddaten noch in ein internes, sogenanntes Verschieberegister. Dort stehen die Bilddaten für den Fall zur Verfügung, daß mehrere Zeilen gleichartig sind (wenn zum Beispiel ein Pixel mehrere Bildzeilen einnimmt). Auf diese Art liest der ANTIC die Bilddaten für gleichartige Zeilen nur einmal und spart dadurch DMA-Zyklen ein.

DIE EINZELNEN MODI:

Die Angaben beziehen sich, wenn nicht anders angegeben, auf die normale Zeilenbreite (160 Farbtakte pro Bildzeile).

ANTIC-MODUS	Beschreibung
"8"	Dieser ANTIC-Modus entspricht dem BASIC-Modus "3". Es werden 40 Pixel pro Normalzeile dargestellt, dabei stehen vier verschiedene Farben zur Verfügung. Jedes Pixel hat eine Höhe von 8 Bildzeilen. Pro Normalzeile werden 10 Bytes Bildspeicher benötigt. Jedes Pixel belegt 4 Farbtakte. Bei normaler Bildschirmeinstellung sind die Pixel also quadratisch. Jeweils zwei Bits der Bilddaten wählen ein Farbregister an, dabei gilt folgende Tabelle:

Daten	Farbregister
00	COLBAK
01	COLPF0
10	COLPF1
11	COLPF2

"9"	Dieser ANTIC-Modus entspricht dem BASIC-Modus "4". Es werden 80 Pixel pro Normalzeile dargestellt, dabei ist jedes Pixel 2 Farbtakte breit. Es stehen zwei verschiedene Farben zur Verfügung. Pro Zeile werden auch bei diesem Modus 10 Byte Bildspeicher benötigt. Auch bei diesem Modus sind die Pixel normalerweise quadratisch. Für jedes Pixel steht ein Bit in den Bilddaten zur Verfügung, wenn das Bit "0" ist, wird die Farbe aus COLBAK (Hintergrund) abgebildet, ansonsten erscheint die Farbe aus COLPF0.

"A" Dieser ANTIC-Modus entspricht dem BASIC-Modus "5". Auch hier sind die Pixel quadratisch. Pro Zeile werden 80 Pixel , die je zwei Farbtakte breit sind, dargestellt. Es stehen vier verschieden Farben für jedes Pixel zur Verfügung. Pro Zeile werden 20 Bytes Bildspeicher benötigt. Jedes Pixel ist 4 Bildzeilen hoch. Je zwei aufeinander folgende Bits der Bilddaten wählen aus vier Farbregistern eins aus, dabei gilt folgende Tabelle:

Daten	Farbregister
00	COLBAK
01	COLPF0
10	COLPF1
11	COLPF2

"B" Dieser ANTIC-Modus entspricht dem BASIC-Modus "6". Pro Zeile werden 160 quadratische Pixel dargestellt, die je einen Farbtakt breit und 2 Bildzeilen hoch sind. Es stehen zwei Farben für die Pixel zur Verfügung. Pro Zeile werden also 20 Byte Bilddaten benötigt. Für die Farbe jedes Pixels ist ein Bit der Bilddaten "zuständig". Wenn dieses Bit "0" ist, wird die Farbe aus COLBAK dargestellt, ansonsten erscheint die Farbe aus COLPF0.

"C" Dieser ANTIC-Modus wird vom Betriebssystem nicht verwendet. Die Pixel sind hier etwa doppelt so breit wie hoch. Pro Zeile werden 160 Pixel dargestellt, die je einen Farbtakt breit und eine Bildzeile hoch sind. Es stehen zwei Farben zur Verfügung. Pro Zeile werden auch hier 20 Bytes Bilddaten benötigt. Für jedes Pixel ist ein Bit der Bilddaten "zuständig". Wenn dieses Bit "0" ist, wird die Farbe aus COLBAK dargestellt, ansonsten erscheint die Farbe aus COLPF0.

"D" Dieser ANTIC-Modus entspricht dem BASIC-Modus "7". Die Pixel sind hier quadratisch. Pro Normalzeile werden 160 Pixel abgebildet, für die vier Farben zur Verfügung stehen. Jedes Pixel ist einen Farbtakt breit und zwei Bildzeilen hoch. Pro Zeile werden also 40 Bytes Bilddaten benötigt. Je zwei aufeinander folgende Bits der Bilddaten wählen aus vier Farbregistern eins aus, dabei gilt folgende Tabelle:

Daten	Farbregister
00	COLBAK
01	COLPF0
10	COLPF1
11	COLPF2

"E" Auch dieser ANTIC-Modus wird nicht vom Betriebssystem bzw. BASIC benutzt. Die Pixel sind auch hier etwa doppelt so breit wie hoch. Pro Zeile werden 160 derartige Pixel dargestellt, für die vier Farben zur Verfügung stehen. Jedes Pixel ist einen Farbtakt breit und eine Bildzeile hoch. Für eine Zeile werden 40 Bytes Bildspeicher benötigt. Bei diesem Modus ist besondere Vorsicht bei der Erstellung eines ANTIC-Programms geboten: Wenn man diesen Modus mit 192 Zeilen bzw. Bildzeilen (hier sind Zeilen und Bildzeilen identisch) betreibt, ist der Bildspeicher größer als 4 KByte, etwa in der Mitte des ANTIC-Programms muß daher der Bildspeicherzähler durch einen entsprechenden ANTIC-Befehl neu geladen werden.
Je zwei aufeinander folgende Bits der Bilddaten wählen aus vier Farbregistern eins aus, dabei gilt die folgende Zuordnung:

Daten	Farbregister
00	COLBAK
01	COLPF0

10	COLPF1
11	COLPF2

"F" Dieser ANTIC-Modus entspricht dem BASIC-Modus "8". In einer Zeile werden 320 quadratische Pixel dargestellt, die eine Bildzeile hoch und einen halben Farbtakt breit sind. Da die Pixel nur noch einen halben Farbtakt breit sind, können sie nur unterschiedliche Helligkeiten und nicht verschiedene Farben haben. Pro Bildzeile werden auch bei diesem Modus 40 Bytes Bildspeicher benötigt. Auch bei der Erstellung eines ANTIC-Programms für diesem Modus ist besondere Vorsicht geboten: Wenn man diesen Modus mit 192 Zeilen bzw. Bildzeilen (hier sind Zeilen und Bildzeilen identisch) betreibt, ist der Bildspeicher größer als 4 KByte, etwa in der Mitte des ANTIC-Programms muß daher der Bildspeicherzähler durch einen entsprechenden ANTIC-Befehl neu geladen werden. Die Farbe der Pixel ist immer die Farbe, die in COLPF2 steht. Für jedes Pixel steht ein Bit in den Bilddaten zur Verfügung. Wenn dieses Bit "0" ist, wird die Helligkeit des Pixels ebenfalls durch COLPF2 bestimmt. Ist das Bit "1" wird das Pixel mit der Helligkeit dargestellt, die in COLPF1 eingetragen ist.

Auf der nächsten Seite folgt eine Tabelle, die die einzelnen Grafik-Modi noch einmal zusammenfaßt.

ANTIC-MODUS	"8"	"9"	"A"	"B"	"C"	"D"	"E"	"F"
Basic-Modus	"3"	"4"	"5"	"6"	---	"7"	---	"8"
Pixel pro Zeile (schmales Bild)	32	64	64	128	128	128	128	256
Pixel pro Zeile (breites Bild)	48	96	96	192	192	192	192	384
Pixel pro Zeile (Normalzeile)	40	80	80	160	160	160	160	320
Bytes pro Zeile (Normalzeile)	10	10	20	20	20	40	40	40
Bildzeilen (pro Pixel)	8	4	4	2	1	2	1	1
Farbtakte (pro Pixel)	4	2	2	1	1	1	1	1/2
Farbanzahl	4	2	4	2	2	4	4	1,5
Datenbits (pro Pixel)	2	1	2	1	1	2	1	1

VERWENDETE FARBREGISTER								
COLPF0	X	X	X	X	X	X	X	
COLPF1	X		X			X	X	H
COLPF2	X		X			X	X	X
COLPF3								
COLBAK	X	X	X	X	X	X	X	

H : Nur die Helligkeitsangabe wird verwendet !

```
**************************
*                        *
*                        *
*   DIE SCHRIFTGRAFIK    *
*   -----------------    *
*                        *
**************************
```

Der ANTIC verfügt nicht nur über leistungsfähige Modi zur Darstellung von Punktgrafiken, er besitzt auch 6 Modi zur Darstellung von Schriftzeichen. Den Begriff Schriftzeichen sollte man allerdings nicht allzu wörtlich nehmen. Natürlich kann man auch andere Gegenstände mit den Schriftgrafik-Modi auf dem Bildschirm darstellen, man muß sich lediglich einen eigenen Zeichensatz definieren. Mehrere Schriftgrafik-Modi eignen sich auch nicht ohne weiteres zur Darstellung von Buchstaben, da sie nur vier Pixel breit sind. Dafür sind sie für andere Zeichendarstellungen (z.B. in Spielen) geeignet, da sie über vielfältige Farbmöglichkeiten verfügen. Mit den Schriftgrafik-Modi kann man sehr hoch aufgelöste Bilder mit wenig Bildspeicherraum darstellen.

Der wesentliche Unterschied zwischen Bild- und Schriftgrafik-Modi liegt in der Quelle der Daten, die auf dem Bildschirm dargestellt werden. Bei der Bildgrafik werden die Daten aus dem Bildspeicher direkt auf dem Bildschirm abgebildet. Bei Schriftgrafik-Modi werden dagegen die Daten aus dem Bildspeicher als Nummer eines Zeichens (als Zeichenname) aus einem Zeichensatz aufgefaßt. Die Bilddaten werden also indirekt über einen Zeichensatz, der irgendwo im Arbeitsspeicher liegt (einen sogenannten "Zeichengenerator"), auf dem Bildschirm dargestellt. Man spricht bei dieser Form der Darstellung auch oft von einer Zeichen-Projektion. Der ANTIC liest dabei die Daten aus dem Bildspeicher und transportiert sie in das Verschieberegister. Je nachdem, welche Bildzeile einer Zeile gerade dargestellt werden soll, liest der ANTIC per DMA mit Hilfe der Daten aus dem Bildspeicher, die er ja im Schieberegister hat, die Bilddaten aus dem Zeichengenerator.

Für jedes Zeichen sind acht Bytes im Zeichengenerator vorhanden. Jedes Byte eines Zeichens bestimmt das Aussehen des Zeichens in einer Zeile von Pixeln (also in ein oder zwei Bildzeilen je nach Modus). Der Zeichengenerator ist eine Liste aller Zeichendaten. Der Zeichengenerator enthält keine Lücken, die Zeichen sind direkt aneinandergereiht.

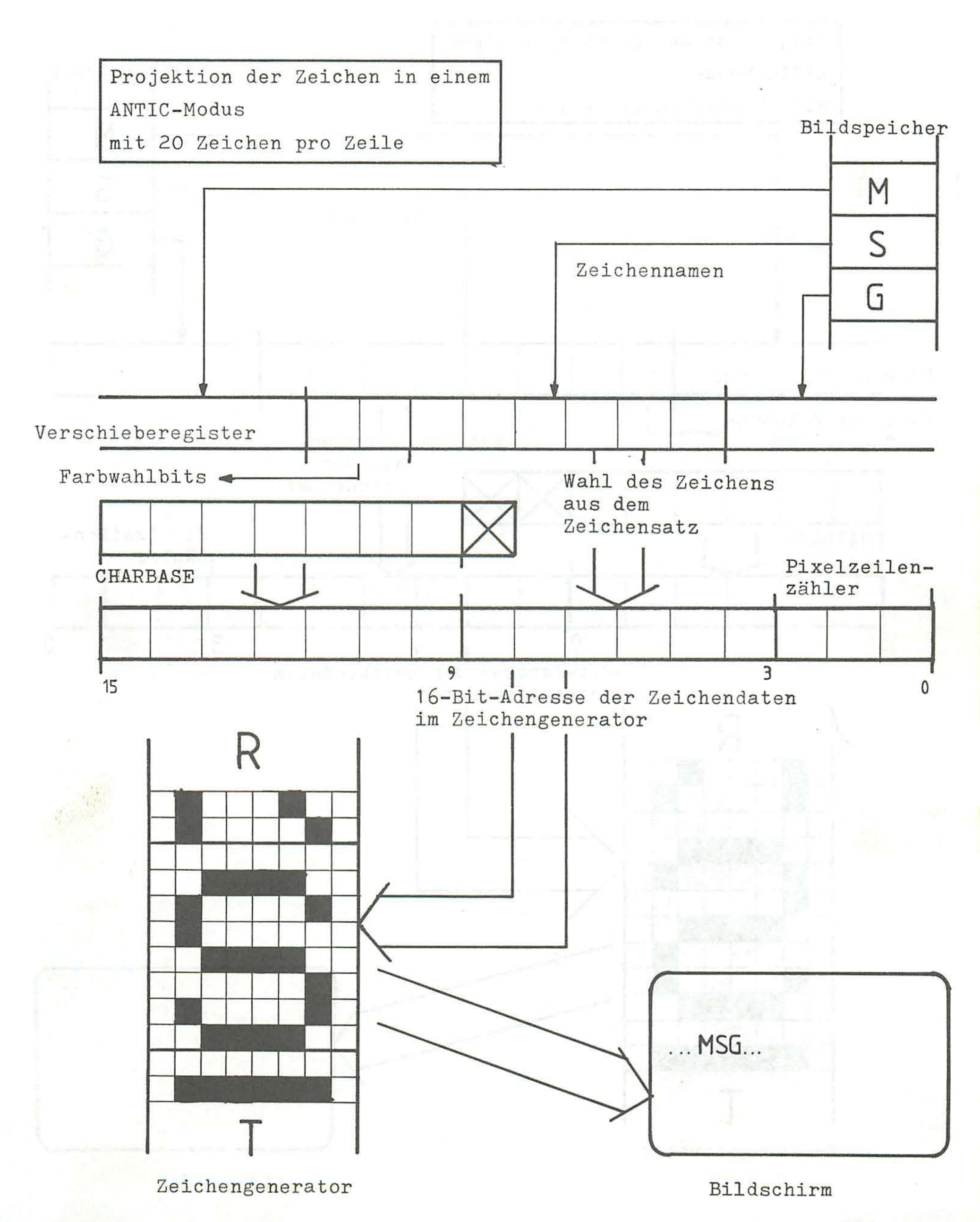
Nach der Projektion einer Pixelzeile (Eine Pixelzeile hat ein oder zwei Bildzeilen, je nachdem, welcher ANTIC-Modus benutzt wird) wird intern im ANTIC der Zeilenzähler um eins erhöht (inkrementiert). Dadurch werden, nachdem zum Beispiel die Daten für die erste Bildzeile aus dem Zeichengenerator gelesen wurden, die Daten für die zweite Bildzeile aus dem Zeichengenerator gelesen.

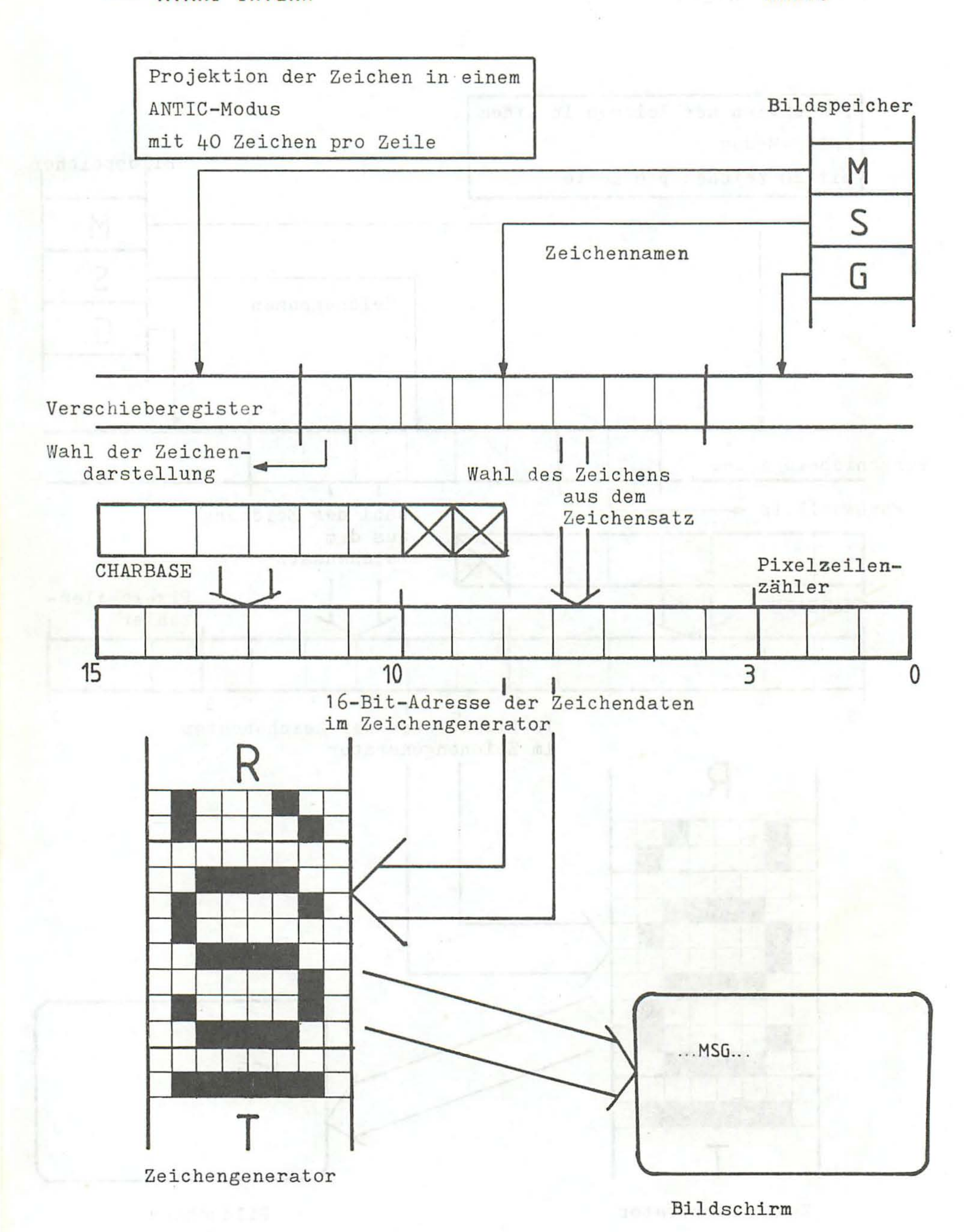
Natürlich muß man dem ANTIC mitteilen, wo er den Zeichengenerator findet. Die Nummer des 256 Byte-Blocks, an dem der Zeichengenerator beginnt, schreibt man in das Register bzw. Schattenregister

CHARBASE	54281	$d409
CHARBASE§	756	$2f4

Bei Modi mit 40 Zeichen pro Zeile werden Bit 0 bis 6 der Bildspeicherdaten als Name des darzustellenden Zeichens aufgefaßt. Der Zeichengenerator umfaßt also 128 Zeichen. Da jedes Zeichen 8 Bytes belegt, werden 1024 Bytes Zeichengenerator benötigt. Bei diesen Modi werden die Bits 2 bis 7 von CHARBASE benutzt. Der Zeichengenerator muß daher an einer 1 KBYTE-Grenze im System beginnen.

Bei Modi mit 20 Zeichen pro Zeile werden Bit 0 bis 5 der Bildspeicherdaten als Name des darzustellenden Zeichens aufgefaßt. Der Zeichengenerator umfaßt also 64 Zeichen. Da jedes Zeichen 8 Bytes belegt, werden 512 Bytes Zeichengenerator benötigt. Bei diesen Modi werden die Bits 1 bis 7 von CHARBASE benutzt. Der Zeichengenerator muß daher an einer 512 Byte-Grenze im System beginnen.


Projektion der Zeichen in einem
ANTIC-Modus
mit 20 Zeichen pro Zeile
Bildspeicher
M
S
G
Zeichennamen
Verschieberegister
Farbwahlbits
Wahl des Zeichens
aus dem
Zeichensatz
CHARBASE
Pixelzeilen-
zähler
15
9
3
0
16-Bit-Adresse der Zeichendaten
im Zeichengenerator
R
T
...MSG...
Zeichengenerator
Bildschirm


Projektion der Zeichen in einem
ANTIC-Modus
mit 40 Zeichen pro Zeile
Bildspeicher
M
S
G
Zeichennamen
Verschieberegister
Wahl der Zeichen-
darstellung
Wahl des Zeichens
aus dem
Zeichensatz
CHARBASE
Pixelzeilen-
zähler
15
10
3
0
16-Bit-Adresse der Zeichendaten
im Zeichengenerator
R
T
Zeichengenerator
...MSG...
Bildschirm

Die untersten 6 (bei Modi mit 64 Zeichen im Zeichensatz) bzw. 7 (bei Modi mit 128 Zeichen) Bits eines Bytes aus dem Bildspeicher bestimmen also, welches Zeichen aus dem Zeichensatz abgebildet wird. Das oberste bzw. die beiden obersten Bits bestimmen bei mehreren Modi die Farbe. Bei zwei Modi (ANTIC-Modi "2" und "3") werden durch das oberste Bit (Bit 7) eines Bytes aus dem Bildspeicher zusätzliche Funktionen gesteuert. Für diese Zusatzfunktionen und die Steuerung der Schriftgrafik-Modi überhaupt, steht das Register bzw. Schattenregister

CHARCNTL	54273	$d401
CHARCNTL§	755	$2f3

zur Verfügung.

Die Bits dieses Registers haben folgende Funktionen:

Bit 0 : Zeichen-Ausblendung:
Dieses Bit hat nur bei den Modi "2" und "3" eine Funktion. Wenn es bei diesen Modi gesetzt ist, veranlaßt es, daß alle Zeichen, bei denen das höchste Bit (Bit 7) gesetzt ist, nicht auf dem Bildschirm erscheinen. Wenn man das Bit 7 von einem Zeichen setzt und Bit 0 von CHARCNTL periodisch umschaltet, blinkt das Zeichen periodisch auf.

Bit 1 : Zeichen-Invertierung:
Dieses Bit hat nur bei den Modi "2" und "3" eine Funktion. Wenn es bei diesen Modi gesetzt ist, veranlaßt es, daß bei allen Zeichen, bei denen Bit 7 (das höchste Bit) gesetzt ist, die Farben von Zeichen und Zeichenhintergrund vertauscht werden. Ein Zeichen, das sonst weiß auf blauem Grund erscheint, wird dann blau auf weißem Grund dargestellt. Diese Funktion wird zur Darstellung des Cursors verwendet.

Bit 2 : Dieses Bit wird am Anfang jeder Schriftzeile abgefragt. Wenn es gesetzt ist, wird die

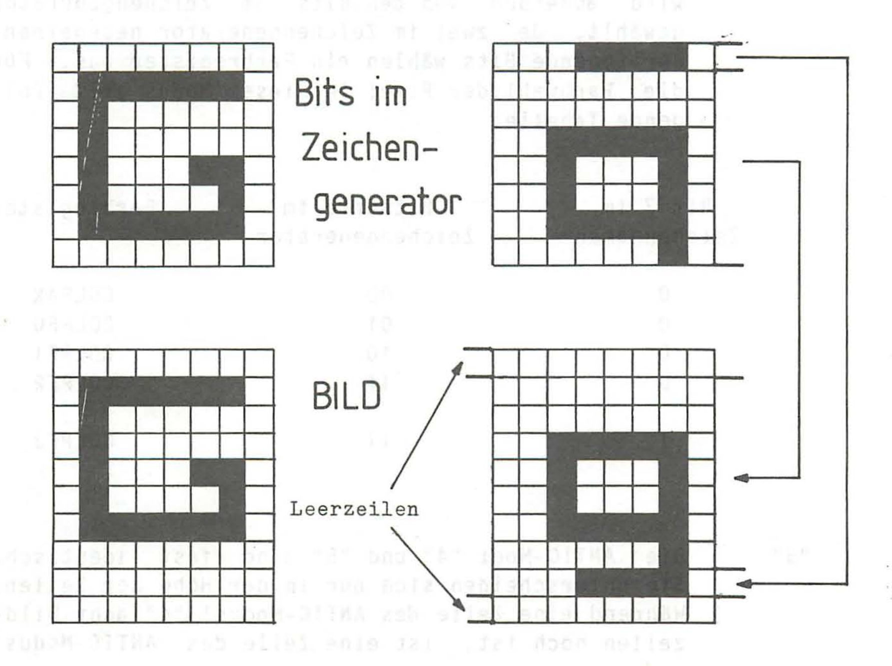
Schriftzeile komplett auf den Kopf gestellt (vertikal gespiegelt).

Bit 3-7 : unbenutzt

Es folgt eine Aufstellung der einzelnen Schriftgrafik-Modi:

ANTIC-MODUS	BESCHREIBUNG
"2"	Dieser ANTIC-Modus entspricht dem BASIC-Modus "0", also dem, der nach dem Einschalten des Atari aktiviert ist. Es werden 40 Zeichen pro Normalzeile dargestellt. Jedes dieser Zeichen besteht horizontal aus 8 Pixeln, die jeweils einen halben Farbtakt breit sind. Dadurch können die einzelnen Pixel auch nicht mehr unterschiedliche Farben, sondern nur unterschiedliche Helligkeiten haben. Eine Zeile ist 8 Pixel hoch, dabei belegt jedes Pixel eine Bildzeile. Jedes Bit der Bilddaten im Zeichengenerator bestimmt die Helligkeit eines Pixels, die Farbe stammt immer aus COLPF2. Wenn das Bit "0" ist, wird auch die Helligkeit aus COLPF2 genommen, ansonsten wird die Helligkeit aus COLPF1 genommen. Der Zeichensatz für diesen Modus besteht aus 128 Zeichen, somit werden 1024 Byte Zeichengenerator benötigt. Bit 7 der Zeichennamen (der Daten im Bildspeicher) steuert das Aussehen des Zeichens über das Register CHARCNTL/CHARCNTL§.
"3"	Dieser ANTIC-Modus wird vom Betriebssystem nicht benutzt. Wie ANTIC-Modus "2", ist er besonders zur Darstellung vom Schriftzeichen geeignet. Der wesentliche Unterschied zum ANTIC-Modus "2" ist, daß es mit diesem Modus möglich ist, die Buchstaben mit Unterlängen darzustellen. Unterlängen werden zum Beispiel für den Buchstaben "g" benötigt: Der

untere Teil des "g" muß unter den anderen Buchstaben liegen. In einer Normalzeile werden 40 Zeichen mit je 8 Pixeln horizontal dargestellt, dabei belegt jedes Pixel einen halben Farbtakt. Nebeneinander liegende Pixel können daher auch bei diesem Modus nur unterschiedliche Helligkeiten haben. Jedes Bit aus dem Zeichengenerator bestimmt dabei die Helligkeit eines Pixels, eine "0" wählt die Helligkeit aus COLPF2, eine "1" wählt die Helligkeit aus COLPF1. Die Farbe stammt immer aus COLPF2. Eine Zeile dieses Modus' ist 10 Pixel hoch, dabei belegt jedes Pixel eine Bildzeile. Auch bei diesem Modus enthält der Zeichensatz 128 Zeichen, das achte Bit der Zeichennamen (Bytes im Bildspeicher) steuert wiederum das Aussehen des Zeichens über Register CHARCNTL. Die folgende Skizze stellt die Darstellung der Zeichen aus dem Zeichengenerator dar:



"4" Auch dieser Modus wird nicht vom BASIC benutzt. In diesem Modus werden wieder 40 Zeichen pro Normalzeile abgebildet. Die einzelnen Zeichen sind hier aber nur 4 Pixel breit, dabei belegt jedes Pixel einen Farbtakt. Dieser Modus eignet sich dadurch besonders für Darstellungen in Spielen, die in vielen Farben dargestellt werden sollen, aber nur wenig Speicherraum benötigen dürfen. Landkarten zum Beipiel, lassen sich mit diesem Modus sehr gut darstellen. Die Zeichen haben eine Höhe von 8 Pixeln, dabei ist jedes Pixel eine Bildzeile hoch. Eine Zeile in diesem Modus ist also 8 Bildzeilen hoch. Die Bits 0 bis 6 der Zeichennamen wählen auch hier, welches Zeichen aus dem Zeichensatz dargestellt wird. Der Zeichensatz umfaßt also auch bei diesem Modus 128 Zeichen. Bit 7 der Zeichennamen (der Daten des Bildspeichers) wird für die Wahl der Farben verwendet. Die Farbe eines Pixels wird außerdem von den Bits im Zeichengenerator gewählt. Je zwei im Zeichengenerator nebeneinanderliegende Bits wählen ein Farbregister aus. Für die Farbwahl der Pixel in diesem Modus gilt folgende Tabelle:

Bit 7 im Zeichennamen	Bitwerte im Zeichengenerator	Farbregister
0	00	COLBAK
0	01	COLPF0
0	10	COLPF1
0	11	COLPF2
1	11	COLPF3

"5" Die ANTIC-Modi "4" und "5" sind fast identisch. Sie unterscheiden sich nur in der Höhe der Zeilen. Während eine Zeile des ANTIC-Modus' "4" acht Bildzeilen hoch ist, ist eine Zeile des ANTIC-Modus'

"5" 16 Bildzeilen hoch. Aber auch im ANTIC-Modus "5" liegen 8 Pixel in einer Zeile übereinander, im Gegensatz zum ANTIC-Modus "4" ist hier aber jedes Pixel zwei Bildzeilen hoch. Auch der ANTIC-Modus "5" wird nicht vom BASIC unterstützt. Die Wahl der Farben ist in den ANTIC-Modi "4" und "5" identisch.

"6" Dieser ANTIC-Modus wird vom BASIC als BASIC-Modus "1" unterstützt. Bei diesem Modus werden 20 Zeichen pro Normalzeile abgebildet. Jedes Zeichen ist acht Pixel breit, ein Pixel belegt einen Farbtakt. Eine Zeile dieses Modus ist acht Pixel hoch, jedes Pixel ist eine Bildzeile hoch. Für die Zeichen stehen insgesamt fünf verschiedene Farben zur Verfügung. Mit den Bits 0 bis 5 der Daten des Bildspeichers (der Zeichennamen) wählt man das Zeichen aus. In diesem Modus besitzt der Zeichengenerator nur 64 Zeichen. Die Bits 6 und 7 wählen ein Farbregister aus:

Bit7	Bit6	Farbregister
0	0	COLPF0
0	1	COLPF1
1	0	COLPF2
1	1	COLPF3

Für jedes Pixel steht im Zeichengenerator ein Bit zur Verfügung. Wenn dieses Bit "0" ist, hat das Pixel die Farbe aus COLBAK. Die Farbe des Farbregisters, das man über Bit 6 und 7 gewählt hat, erscheint überall dort im Zeichen, wo das entsprechende Bit des Zeichengenerators "1" ist. Dieser ANTIC-Modus eignet sich besonders zur Darstellung großer Schriften, also zum Beispiel für die Schaufensterwerbung. Die Technik der Farbwahl ermöglicht es, verschiedenfarbige Buchstaben mit

dem gleichen Zeichengenerator zu erzeugen. Wer allerdings in einem Zeichen mehr als zwei Farben benötigt, muß die ANTIC-Gänge "4" oder "5" verwenden. Für "normale" Schriften kann man ohne weiteres den normalen Zeichensatz des Betriebssystems verwenden.

"7" Dieser ANTIC-Modus entspricht dem BASIC-Modus "2". Die ANTIC-Modi "6" und "7" sind fast identisch. Sie unterscheiden sich lediglich in der Höhe der Zeilen. Während die Zeilen von ANTIC-Modus "6" acht Bildzeilen hoch sind, jedes Pixel also eine Bildzeile hoch ist, sind die Zeilen, die mit ANTIC-Modus "7" erzeugt werden, 16 Bildzeilen hoch. Jedes Pixel ist hier zwei Bildzeilen hoch. Damit eignet sich dieser Modus hervorragend für Überschriften und für die Werbung. Auch die Wahl der Farben für die Zeichen und den Hintergrund erfolgt im ANTIC-Modus "7" wie im ANTIC-Modus "6". Der Zeichengenerator kann ebenfalls übernommen werden.

Auf der nächsten Seite folgt eine Übersicht über die ANTIC-Schriftgrafik-Modi.

ANTIC-Modus	"2"	"3"	"4"	"5"	"6"	"7"
Basic-Modus	"0"	---	---	---	"1"	"2"
Anzahl der Zeichen pro Zeile	40	40	40	40	20	20
Anzahl der Pixel pro Zeichen (horizont.)	8	8	4	4	8	8
Anzahl der Farbtakte pro Pixel (horizont.)	1/2	1/2	1	1	1	1
Anzahl der Pixel pro Zeichen (vertikal)	8	8	8	8	8	8
Anzahl der Bildzeilen pro Zeile	8	10	8	16	8	16
Anzahl der Bildzeilen pro Pixel	1	1	1	2	1	2
Anzahl der Farben	1,5	1,5	5	5	5	5
Anzahl der Buchstaben pro Zeichensatz	128	128	128	128	64	64
Anzahl der Bytes pro Zeichensatz	1K	1K	1K	1K	512	512
Anzahl der Bytes im Bildspeicher pro Zeile	40	40	40	40	20	20
Anzahl der Datenbits aus dem Zeichengenerator pro Pixel	1	1	2	2	1	1

```
*****************************************
*                                       *
*                                       *
*   SCROLLING (VERSCHIEBEN DES BILDES)  *
*   ---------------------------------   *
*                                       *
*****************************************
```

Häufig existiert das Problem, mehr auf dem Bildschirm darstellen zu müssen, als Platz findet. Eine Möglichkeit wäre, das gesamte Bild nach bestimmten Tastenkommandos zu wechseln, eine andere automatisch im Programmablauf zwischen den verschiedenen Bildern umzuschalten.

Die meisten Mikrocomputern bieten höchstens die Auswahl zwischen zwei Bildspeichern und somit zwei Bildern. Möchte man zwischen mehr Bildern umschalten oder, wenn nur ein Bildspeicher vorhanden ist, überhaupt ein zweites Bild anzeigen, muß der gesamte Inhalt des Bildspeichers von der CPU verschoben werden. Da dies für die CPU sehr arbeitsintensiv ist, geschieht es zu langsam, Störungen des Bildes sind kaum zu vermeiden. Der Atari bietet dagegen eine komfortable Möglichkeit, das Bild zu wechseln. Man braucht lediglich die Adresse des ANTIC-Programms zu wechseln. Wenn das ANTIC-Programm für beide Bilder identisch ist, reicht es aus, im ANTIC-Programm den Befehl, der die Adresse des Bildspeichers in den Bildspeicherzählers schreibt, jeweils abzuändern.

Gerade bei der Bearbeitung großer Tabellen oder beim Schreiben von Texten mit mehr als 40 Zeichen pro Zeile, wird jedoch ein Bildwechsel als störend empfunden.

Eine weitaus bessere Möglichkeit, mehr auf dem Bildschirm darzustellen, als auf einmal auf ihn paßt, ist es, das Bild vertikal oder horizotal in kleinen Schritten zu verschieben. Der Bildschirm fungiert nur noch als Fenster, durch das man ein größeres Feld betrachtet. Auf diese Art entsteht der Eindruck, auf einem großen Bild zu arbeiten.

Auch dies ist beim Atari leichter möglich als bei anderen Mikrocomputern. Bei vielen anderen Geräten muß der gesamte Bildspeicherinhalt verschoben werden. Dabei ist der Rechenaufwand der CPU enorm. Ein sich nur langsam verschiebendes Bild ist die Folge. Der Atari ermöglich es, für jede Zeile den Bildspeicherzähler neu durch das ANTIC-Programm laden zu lassen. Es können auch mehr Zeilen, als auf den Bildschirm passen, bereits im Speicher vorbereitet werden (ein Bildspeicher mit mehr Zeilen, als auf den Bildschirm passen, wird dazu angelegt). Soll das Bild verschoben werden, braucht man nur das ANTIC-Programm entsprechend zu ändern. Die Änderung des ANTIC-Programms erfordert weniger Rechenaufwand als die Verschiebung des gesamten Bildspeicherinhalts.

BEISPIEL:

Der Bildschirm ist in zwei Teile zu unterteilen. Der obere Teil soll auf dem Bilschirm 16 Zeilen mit je 40 normalen Buchstaben umfassen, sich aber über ein Feld von 32 * 256 Zeichen bewegen lassen. Der untere Teil des Bildschirm soll 8 feststehende Textzeilen umfassen. Das Programm, das diesen Bildschirm benötigt, läßt den Arbeitsspeicher ab Adresse $5000 frei.

Das folgende ANTIC-Programm liefert ein derartiges Bild:
(alle Daten und Adressen hexadezimal):

ADRESSE	DATEN	KOMMENTAR
-------	-----	---------
	das ANTIC-Programm für das Bild:	
5000	70	
	70	24 Leerzeilen
	70	
5003	42	1. Bildzeile
	XPOS	Bildspeicherzähler wird
	YPOS + $60 + 0	geladen
	42	2. Bildzeile
	XPOS	Bildspeicherzähler wird
	YPOS + $60 + 1	geladen
	42	3. Bildzeile
	XPOS	Bildspeicherzähler wird
	YPOS + $60 + 2	geladen
	42	4. Bildzeile
	XPOS	Bildspeicherzähler wird
	YPOS + $60 + 3	geladen
	.	
	.	
	.	
	.	
	.	
	.	
	.	
	42	15. Bildzeile
	XPOS	Bildspeicherzähler wird
	YPOS + $60 + $e	geladen
	42	16. Bildzeile
	XPOS	Bildspeicherzähler wird
	YPOS + $60 + $f	geladen
5030	42	1. Zeile des
	00	feststehenden Bildteils
	55	Bildspeicherzähler laden
	02	2. Zeile
	02	3. Zeile

```
                 .                  .
                 .                  .
                 .                  .
                 .                  .
                 02                 7. Zeile
                 02                 8. Zeile
503a             41                 springe nach $5000 und
:IQ?auf das Ende der
                 50                 Vertikalsynchronisation

    hier endet das ANTIC-Programm

5500             Beginn des Bildspeichers des feststehenden
                 Bildteils (1. Zeile)
5528             2. Zeile des feststehenden Bildteils
5550             3. Zeile des feststehenden Bildteils
5578             4. Zeile des feststehenden Bildteils
55a0             5. Zeile des feststehenden Bildteils
55c8             6. Zeile des feststehenden Bildteils
55f0             7. Zeile des feststehenden Bildteils
5618             8. Zeile des feststehenden Bildteils
563f             ENDE des Bildspeichers des feststehenden
                 Bildteils

6000             Beginn des Feldes (mit 32 * 256 Zeichen) für
                 den verschiebbaren Teil des Bildes
                 1. Zeile des Feldes
6100             2. Zeile des Feldes
6200             3. Zeile des Feldes
6300             4. Zeile des Feldes
6400             5. Zeile des Feldes
6500             6. Zeile des Feldes
.                .
.                .
.                .
7d00             31. Zeile des Feldes
7e00             32. Zeile des Feldes
7fff             ENDE des Feldes für den verschiebbaren Teil
                 des Bildes
```

Dieses Beispiel ist bewußt sehr einfach aufgebaut. Bei jeder Änderung der "Position" des Bildes über dem Speicherfeld muß das ANTIC-Programm geändert werden. Es soll dabei nach dem im Beispiel angegebenen Muster erstellt werden. Für XPOS wird die horizontale Position im Speicherfeld eingesetzt, für YPOS die vertikale. Es muß darauf geachtet werden, daß XPOS nie größer als $d8 und YPOS nie größer als $10 wird. Ansonsten treten Fehler auf, weil die Zeilenenden des Bildschirms die Zeilenenden des Speicherfeldes überschreiten. Das Beispiel nutzt den Speicher auch nicht besonders gut aus (es hinterläßt große Lücken zwischen den einzelnen Speicherfeldern), dafür ergeben sich bei der Berechnung der Adressen aber meist "glatte" Werte.

Natürlich kann man dieses Beispiel auch auf hochauflösende ANTIC-Modi übertragen. Allerdings wird dann sehr viel Arbeitsspeicher für das Bildspeicherfeld benötigt. Das Verschieben von Bildteilen "lohnt" sich am meisten bei ANTIC-Modi, die wenig Speicherraum benötigen.

Einen Nachteil hat die beschriebene Methode jedoch. Man erhält kein "pixelweise" ruhig abrollendes Bild, der Bildschirm "springt" immer horizontal um ein Zeichen und vertikal um eine Zeile.

Der ANTIC bietet jedoch sehr komfortable Möglichkeiten zur Verschiebung von Bildteilen oder ganzen Bildern. Es ist mit dem ANTIC möglich, das Bild horizontal um einzelne Farbtakte und vertikal um einzelne Bildzeilen zu verschieben. Dabei bleiben die Spieler und Geschosse jedoch auf ihrer alten Position stehen. Sie werden durch die Verschiebungen nicht beeinflußt.

Im ANTIC stehen zwei Register zur Verfügung, die die Feinverschiebung des Bildes oder eines Bildteiles in horizontaler und vertikaler Richtung steuern. Für die horizontale Verschiebung, das horizontale Scrolling, ist Register

HSCROL 54276 $d404

zuständig. Die vertikale Verschiebung, das vertikale Scrolling, wird von Register

VSCROL 54277 $d405

gesteuert.

Von diesen Registern werden jedoch nicht unbedingt alle Bildzeilen beeinflußt. In jedem ANTIC-Befehl, der eine Erzeugung von Bildzeilen zur Folge hat (also nicht in Leerzeilen-Befehlen), stehen zwei Bits zur Verfügung, die die horizontale bzw. die vertikale Verschiebung der von diesem Befehl erzeugten Bildzeilen einschalten. Auf diese Art ist es möglich, entweder das ganze Bild oder auch nur einzelne Teile des Bildes zu verschieben. Die horizontale Verschiebung wird von

Bit 4

des ANTIC-Befehls eingeschaltet. Für das Einschalten der vertikalen Verschiebung ist

Bit 5

des ANTIC-Befehls zuständig. Hierbei muß der Teil des Bildes, der horizontal verschoben wird, nicht mit dem identisch sein, der vertikal verschoben wird.

Die vertikale Verschiebung

Bei einem Grafik-Modus, der mit jedem ANTIC-Befehl nur eine Bildzeile erzeugt, kann mit der im Beispiel verwendeten Methode das Bild um einzelne Bildzeilen verschoben werden. Bei anderen Modi, zum Beispiel bei Schriftgrafik-Modi, erzeugt jeder ANTIC-Befehl gleich mehrere Bildzeilen. Hier

ergibt sich das Problem, daß mit der im Beispiel verwendeten Methode das Bild nicht um einzelne Zeilen verschiebbar ist.

Der ANTIC bietet mit dem VSCROL-Register die Möglichkeit, auch in ANTIC-Modi, die mehrere Bildzeilen erzeugen, das Bild um einzelne Bildzeilen zu verschieben. Es werden alle Bildzeilen, bei denen Bit 5 im ANTIC-Befehl gesetzt ist, um die Anzahl von Bildzeilen nach oben verschoben, die in VSCROL steht. Die letzte Zeile, die derartig verschoben wird, ist die erste Zeile, bei der Bit 5 des ANTIC-Befehls nicht gesetzt ist. Um zu verstehen, wie der ANTIC die Verschiebung erreicht, muß man sich zuerst den normalen Vorgang der Erzeugung der Bildzeilen vor Augen führen:

Der ANTIC besitzt intern einen Zähler, der kontrolliert, welche Bildzeile der ANTIC-Programm-Zeile gerade abgebildet wird. Dieser Zähler wird in der Literatur auch oft als DCTR bezeichnet. DCTR steht dabei für "Deltacontrol", das Wort "Delta" wird in den meisten Naturwissenschaften für "Differenz" verwendet. Dieser Zähler zählt die Bildzeilen von 0 bis zu einer höchsten Zahl, die durch den ANTIC-Modus bestimmt wird. Diese höchste Zahl ist immer um eins niedriger als die Anzahl der Bildzeilen, die mit dem jeweiligen ANTIC-Modus erzeugt werden. Die folgende Tabelle soll noch einmal den Zählerhöchststand (das Maximum von DCTR) der einzelnen ANTIC-Modi zeigen:

ANTIC-Modus	Zählerhöchststand
"2"	7
"3"	9
"4"	7
"5"	15
"6"	7
"7"	15

"8"	7
"9"	3
"A"	3
"B"	1
"C"	0
"D"	1
"E"	0
"F"	0

Bei ANTIC-Modi, bei denen nur eine Bildzeile erzeugt wird, zählt der DCTR-Zähler nicht. Daher ist der Zählerhöchststand in diesen Modi auch "0". Bei diesen Modi ist eine Verschiebung des Bildes um einzelne Bildzeilen auch ohne Benutzung von VSCROL möglich.

Gewöhnlich wird durch das ANTIC-Programm ein Block von Zeilen definiert, der verschiebbar ist. Bei der Darstellung der ersten Zeile dieses verschiebbaren Blocks (der ersten Zeile, bei der Bit 5 des ANTIC-Befehls gesetzt ist), fängt DCTR nicht wie üblich mit "0" an zu zählen, sondern mit dem Wert, der in VSCROL steht. Dadurch fehlen Bildzeilen im oberen Teil der Zeile. Die Zeile ist nach oben gerückt worden.

Alle weiteren Zeilen, bei denen Bit 5 des ANTIC-Befehls gesetzt ist, werden direkt an die letzte Bildzeile der jeweils vorhergehenden Zeile angefügt. Sie werden normal dargestellt. Da bei der ersten Zeile der verschiebbaren Zeilen oben Bildzeilen fehlen, ist der gesamte Block nach oben verschoben.

Bei der ersten Zeile, bei der Bit 5 im ANTIC-Befehl nicht mehr gesetzt ist, beginnt der Zähler DCTR zwar bei "0", zählt aber nur bis zu dem Wert der in VSCROL steht. Dadurch erscheint diese Zeile als von unten gekürzt. Die erste Zeile, bei der Bit 5 des ANTIC-Befehls nicht mehr gesetzt ist, ist also die letzte Zeile, die nach oben verschoben wird.

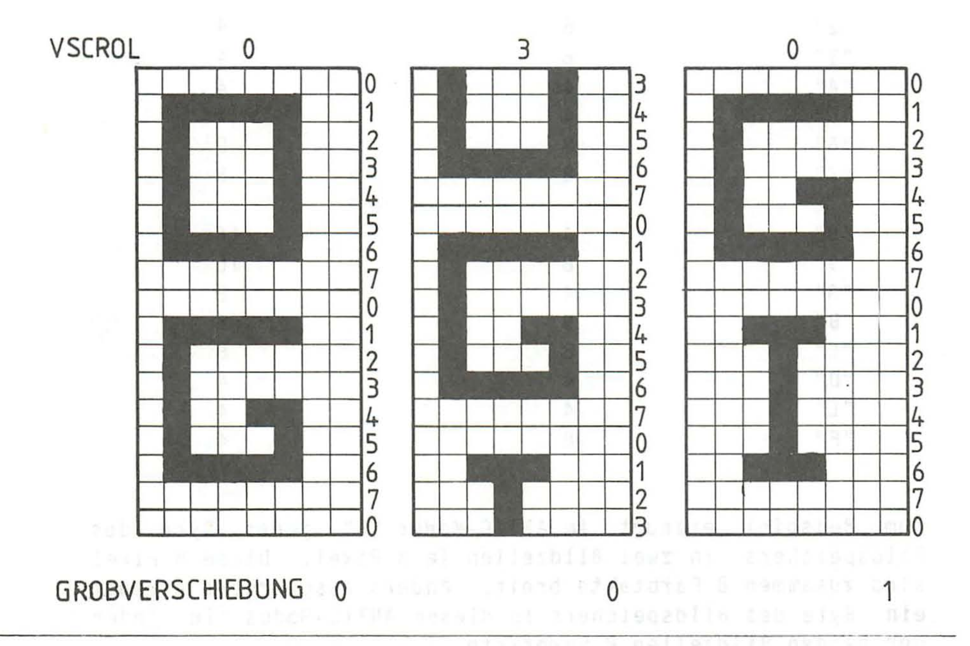
Es wäre unklug, den Wert im Register VSCROL (also die Anzahl der Bildzeilen, um die ein Bildblock nach oben gerückt werden soll), größer zu wählen, als den Maximalstand des DCTR-Zählers. Meistens werden die "überflüssigen" Bits einfach ignoriert, aber gerade bei ANTIC-Modus "3" kann es auch zu anderen Störungen kommen.

Soll der gesamte Bildschirm oder nur die untersten Zeilen verschiebbar gemacht werden, so muß eine weitere Sache beachtet werden: Wenn auch bei der letzten Zeile des Bildes Bit 5 des dazugehörigen ANTIC-Befehls gesetzt ist, rollt diese Zeile nicht in das Bild, sondern springt hinein. Der ANTIC kann nicht erkennen, daß es sich bei dieser Zeile um die letzte zu verschiebende Zeile handelt. Damit der ANTIC auch diese Zeile richtig behandelt, muß bei ihr Bit 5 des ANTIC-Befehls gelöscht werden. Jetzt kann der ANTIC erkennen, daß dies die letzte zu verschiebende Zeile ist und er "rollt" auch diese Zeile ins Bild.

Mit dem VSCROL-Register kann man das Bild natürlich nicht um mehr als eine Zeile, das heißt um eine bis 15 Bildzeilen, abrollen lassen. Meist möchte man das Bild aber über eine sehr viel größere Fläche bewegen. Um dies zu erreichen, muß die Technik der groben Verschiebung, die vorher im Beispiel verwendet wurde, mit der Feinverschiebung mit dem VSCROL-Register kombiniert werden. Beim Aufwärtsrollen eines normalen Textes, bei dem jeder ANTIC-Befehl 8 Bildzeilen erzeugt (zum Beispiel ANTIC-Modus "2"), sollte man das Bild erst mit 7 Feinverschiebungen nach oben rollen. Statt einer (ohnehin nicht möglichen) achten Feinverschiebung, wird eine Grobverschiebung um eine Zeile vorgenommen und VSCROL wieder auf "0" gesetzt. Beim Abwärtsrollen wird dagegen zuerst die Grobverschiebung um eine Zeile vorgenommen. Gleichzeitig mit dieser Grobverschiebung, wird aber VSCROL auf "7" gesetzt. Die Verschiebungen um die nächsten sieben Bildzeilen sind reine Feinverschiebungen mit VSCROL.

Auf diese Art erhält man ein Bild, das über eine Fläche, die nur durch den verfügbaren Speicherraum begrenzt wird, um einzelne Bildzeilen verschoben werden kann.

Die folgende Grafik soll das Prinzip der vertikalen Verschiebung demonstrieren:



Die horizontale Verschiebung

Im Beispiel wurde gezeigt, wie das Bild um jeweils ganze Bytes des Bildspeichers horizontal verschoben werden kann. Bei der Verschiebung um ganze Bytes des Bildspeichers springt das Bild jedoch, da jedes Byte des Bildspeichers, je nach ANTIC-Modus, unterschiedlich viele Pixel in einer Bildzeile, also auch unterschiedlich viele Farbtakte, erzeugt.

Die folgende Tabelle zeigt noch einmal, welcher ANTIC-Modus in einer Bildzeile wieviele Farbtakte und Pixel pro Byte des Bildspeichers erzeugt:

ANTIC-MODUS	Anzahl der Pixel pro Byte des Bildspeichers	Anzahl der Farbtakte und Bildzeile
"2"	8	4
"3"	8	4
"4"	4	4
"5"	4	4
"6"	8	8
"7"	8	8
"8"	4	16
"9"	8	16
"A"	4	8
"B"	8	8
"C"	8	8
"D"	4	4
"E"	4	4
"F"	8	4

Zum Beispiel erzeugt im ANTIC-Modus "B" jedes Byte des Bildspeichers in zwei Bildzeilen je 8 Pixel. Diese 8 Pixel sind zusammen 8 Farbtakte breit. Anders ausgedrückt erzeugt ein Byte des Bildspeichers in diesem ANTIC-Modus in jeder der beiden Bildzeilen 8 Farbtakte.

Der ANTIC bietet mit dem Register HSCROL eine Möglichkeit, das Bild auch um einzelne Farbtakte zu verschieben. Er verschiebt die Bildzeilen, bei deren ANTIC-Befehl das Bit 4 gesetzt ist, um die Anzahl von Farbtakten nach rechts, die in Register HSCROL steht. Dabei sind nur die untersten 4 Bits von HSCROL gültig. Das Bild läßt sich also nur um maximal 15 Farbtakte verschieben. Für darüber hinausgehende

Verschiebungen gilt, wie beim vertikalen Verschieben, daß diese Art der "Feinverschiebung" mit der anfangs gezeigten Grobverschiebung kombiniert werden muß.

Wenn der ANTIC einen Befehl liest, bei dem Bit 4 gesetzt ist, unterdrückt er eine, dem Inhalt von HSCROL entsprechende, Anzahl von Farbtakten (und somit Pixeln) am Anfang jeder, zu diesem ANTIC-Befehl gehörigen, Bildzeile.

Der ANTIC benötigt zur Darstellung von horizontal verschobenen Zeilen mehr Daten, als bei der Darstellung von Normalzeilen, da ja zum Beispiel bei Textmodi mit 40 Zeichen pro Zeile, links das 41. Zeichen auftaucht.

Daher holt der ANTIC sich, wenn im Register DMACNTL das schmale Spielfeld gewählt wurde, pro Zeile die Anzahl von Bytes aus dem Speicher, die sonst beim normalen Spielfeld pro Zeile verwendet werden.

Bei normaler Spielfeldbreite liest der ANTIC die Anzahl von Bytes, die er sonst beim breiten Spielfeld lesen würde.

Beim breiten Spielfeld kommt kein Wechsel der Byteanzahl in Frage, es wird Hintergrundfarbe eingeschoben.

Normalerweise wird man bei Verwendung der horizontalen Feinverschiebung ohnehin bei jedem ANTIC-Befehl den Bildspeicherzähler neu setzen (wie auch im Beispiel). Daher ist es nur bedingt interessant, wieviele Bytes der ANTIC tatsächlich für eine Zeile liest. Man muß lediglich darauf achten, daß der ANTIC keine Bytes aus Speicherstellen, die hinter dem eigentlichen Zeilenende liegen, liest. Außerdem ist wichtig, daß bei horizontaler Verschiebung am Ende einer Zeile meist ein weiteres Zeichen auftaucht (bei Modi mit 40 Zeichen pro Zeile das 41. Zeichen).

D E R   G T I A

1) ALLGEMEINES

2) DARSTELLUNG DER ANTIC-BILDDATEN

3) DIE PLAYER-MISSILE-GRAFIK

4) DIE TRIGGEREINGÄNGE UND KONSOLENTASTER

Der GTIA ist ein weiterer Spezialbaustein des Atari. Der Begriff ist eine Abkürzung für "Graphic Television Interface Adaptor". Der GTIA ist Hauptverbindungsglied zwischen ANTIC und dem Fernseher.

Die vorrangige Aufgabe des GTIA ist es also, die vom ANTIC übergebenen Daten in ein Fernsehbild umzusetzen. Der GTIA verfügt dazu über Farbregister. In jedes dieser Register wird jeweils eine bestimmte Farbe mit einer bestimmten Helligkeit eingetragen. Wenn der ANTIC ein Bild erstellt, gibt er nicht selbst die Farbe an, sondern wählt nur ein Farbregister des GTIA aus. Dies ist die normale Art der Bilddarstellung, es ist aber auch möglich, den GTIA dazu zu bringen, die vom ANTIC übergebenen Daten anders zu interpretieren.

Eine weitere Aufgabe des GTIA ist die Erzeugung von beweglichen Objekten auf dem Bildschirm, sogenannten "PLAYERn" und "MISSILEs" (Spielern und Geschossen). Diese Form der Darstellung wird daher auch als Player-Missile-Grafik bezeichnet. Es ist möglich, gleichzeitig vier Spieler und vier Geschosse darzustellen. Unter Umständen ist aber auch eine höhere Anzahl zu erreichen. Der GTIA unterstützt bei der Player-Missile-Grafik jeweils nur eine Bildzeile. Das heißt, daß für jede Zeile dem GTIA neue Daten über die Form der Spieler und Geschosse mitgeteilt werden müssen. Wie schon im Kapitel über den ANTIC beschrieben, ist dies auch automatisch durch den DMA möglich.

Für die Player-Missile-Grafik stehen im GTIA Register zur Verfügung. So sind wie für die Spielfelddaten, die der ANTIC übergibt, auch für die Player-Missile-Grafik Farbregister vorhanden. Außerdem gibt es Register für die Horizontalposition der Spieler und Geschosse und deren Aussehen (das heißt, wie ein Spieler bzw. ein Geschoß in EINER Zeile aussieht). Die Register, in denen das Aussehen der Spieler und Geschosse gespeichert wird, heißen Spieler- bzw. Geschoßgrafikregister. Oft werden diese Register in der Literatur auch als "Umrißregister" bezeichnet. Diese Bezeichnung halten wir jedoch für mißverständlich. Auch die horizontale

Größe der Spieler und Geschosse ist in Registern gespeichert und somit programmierbar.

Da das Bild vom ANTIC und die Daten der Spieler- und Geschoßgrafikregister im GTIA gemischt werden, kommt es bei Objekten und Bildteilen, die die gleiche Bildposition belegen zur Frage, was dargestellt werden soll, also: Was im Vorder- und was im Hintergrund liegen soll. Dies läßt sich mit einem Register steuern, das festlegt, welche Objekte bzw. welche Bildschirmfarben an welcher Stelle in einer "Prioritätsliste" stehen.

Besonders bei Spielen ist es auch interssant, ob Spieler oder Geschosse mit Spielern, Geschossen oder Bildschirmteilen zusammengestoßen sind. Auch dies wird dem Benutzer vom GTIA in Registern angezeigt. Durch diese Funktion wird der CPU erheblich Rechenzeit erspart, da sie nicht selbst anhand der Positionangaben ausrechnen muß, ob es "Zusammenstöße" gab.

Zusätzlich zu den beschriebenen Funktionen, ist der GTIA auch für die sogenannten Triggerleitungen zuständig. Die Triggerleitungen sind die Leitungen, die mit den "Joystick-Feuertasten" verbunden sind. Der Status dieser Triggerleitungen wird über GTIA Register abgefragt. Bei den alten Atari-Geräten sind alle vier Triggereingänge des GTIA benutzt, da ja auch vier Joystickports zur Verfügung stehen. Bei den neuen Atari-Modellen sind zwei Triggereingänge mit Joystickports und ein Triggereingang mit einer Freigabe für den Cartridge-Steckplatz verbunden. Der vierte Triggereingang ist unbenutzt.

Auch die Zusatztasten der Tastatur, die sogenannten Konsolentaster, sind mit dem GTIA verbunden. Das bedeutet, daß die Tasten "START", "SELECT" und "OPTION" über den GTIA abgefragt werden.

Das "Klicken", das ertönt, wenn man eine Taste betätigt, wird ebenfalls vom GTIA erzeugt. Bei den alten Atari-Geräten

wird es über einen eingebauten Lautsprecher wiedergegeben. Dieser Lautsprecher wurde bei den neuen Modellen eingespart. Das "Klicken" wird jetzt direkt in das Tonsignal, das sonst vom POKEY über den Modulator zum Fernseher übertragen wird, gemischt.

Viele Adressen sind mit zwei Registern des GTIA belegt. Jeweils ein Register ist dann für eine Funktion zuständig, die nur ein Schreiben in das Register erfordert und ein anderes ist für eine Funktion zuständig, die nur ein Lesen des Registers erfordert. So können beide Register die gleiche Adresse haben und sich trotzdem nicht gegenseitig stören. Zum Beispiel liegen die Register, die die Horizontalposition der Spieler festlegen, mit denen, die eine Überprüfung, ob Kollisionen zwischen den Geschossen und dem Spielfeld vorliegen, ermöglichen, auf den gleichen Adressen.

Auch für viele der GTIA-Register stellt das Betriebssystem Schattenregister zur Verfügung. Jeweils während der Vertikalsynchronisation werden die Register- und Schattenregisterinhalte aktualisiert (bzw. aufeinander angeglichen). Dies ist zum Beispiel zu beachten, wenn man die Farben des Spielfeldes ändern möchte. Wenn man die neuen Informationen in die Register im GTIA schreibt, die ein Schattenregister haben, so werden sie bei der nächsten Vertikalsynchronisation überschrieben, tauchen also maximal eine Fünfzigstelsekunde auf dem Bildschirm auf. Um die neue Farbe dauerhaft auf den Bildschirm zu bringen, muß man in das Schattenregister schreiben, da das Betriebssystem von dort aus die Daten liest, die es in das GTIA-Register schreibt. Eine weitere Fehlerquelle liegt oft in der Tatsache, daß die Schattenregisterinhalte erst nach spätestens einer Fünfzigstelsekunde richtig sind. Unter Umständen tritt also der Fall ein, daß ein GTIA-Register und ein Schattenregister zusammen eine unmögliche Aussage machen. Dies führt zu Fehlern in Programmen. Der Benutzer sollte die Informationen also nur aus GTIA- oder nur aus Schattenregistern lesen.

```
********************************************
*                                          *
*                                          *
*  DARSTELLUNG DER BILDATEN DES ANTIC      *
*  ---------------------------------       *
*                                          *
********************************************
```

Wie schon erwähnt, überträgt der ANTIC Daten, die dem GTIA miteilen, welche Farbregister er jeweils abbilden soll. Im Kapitel über den ANTIC war auch schon bei der Beschreibung der diversen Grafikmodi angegeben, welches Bitmuster in den Bilddaten stehen muß, um die Farbe und Helligkeit eines bestimmten Farbregisters auf den Bildschirm zu bringen. Bei Grafikmodi, die eine Auflösung von zwei Bildpunkten pro Farbtakt haben, können benachbarte Punkte nur unterschiedliche Helligkeiten aufweisen, nicht jedoch unterschiedliche Farben. Pro Farbtakt kann nur eine Farbe dargestellt werden.

Alle Farbregister im GTIA sind gleichartig aufgebaut:

Bit 0	unbenutzt
Bit 1-3	Helligkeit
Bit 4-7	Farbe

Die Helligkeit läßt sich also in acht Stufen einstellen (da für sie 3 Bits zur Verfügung stehen). Dabei gilt folgende Tabelle:

Bit3	Bit2	Bit1	Helligkeit
0	0	0	Helligkeit 0, minimal (schwarz)
0	0	1	Helligkeit 1
0	1	0	Helligkeit 2
0	1	1	Helligkeit 3
1	0	0	Helligkeit 4
1	0	1	Helligkeit 5
1	1	0	Helligkeit 6
1	1	1	Helligkeit 7, maximal (weiß)

Es stehen 16 verschiedene Grundfarben zur Verfügung. Allerdings hängt es sehr vom Fernseher ab, wie welche Farbe "wirkt". Bei schlecht eingestellten Fernsehern können auch statt weißen Punkten dicht beieinander liegende andersfarbige Punkte erscheinen. Einen derartigen Fehler, der meist an einer schlechten "Konvergenzeinstellung" des Fernsehers liegt, sollte man vom Fachmann beseitigen lassen.

Für die Wahl der Farben gilt folgende Tabelle:

Bit7	Bit6	Bit5	Bit4	Farbe
0	0	0	0	Grau
0	0	0	1	Gold
0	0	1	0	Orange
0	0	1	1	Rot-Orange
0	1	0	0	Rosa
0	1	0	1	Purpur
0	1	1	0	Violett
0	1	1	1	Blau 1
1	0	0	0	Blau 2
1	0	0	1	Hellblau
1	0	1	0	Türkis
1	0	1	1	Blaugrün
1	1	0	0	Grün
1	1	0	1	Gelbgrün
1	1	1	0	Grün-Orange
1	1	1	1	Hellorange

Für die Spielfeldfarben gibt es vier Register, die mit COLPF0, COLPF1, COLPF2 und COLPF3 bezeichnet werden. Die Abkürzung "COLPF" hat sich in der Literatur eingebürgert, sie steht für "COLOR OF PLAYFIELD", was man mit "SPIELFELDFARBE" übersetzen kann. Außerdem gibt es noch ein Farbregister COLBAK, was für "COLOR OF BACKGROUND" (Hintergrundfarbe) steht. Die Farbe dieses Registers erscheint in jedem Fall überall dort, wo sich kein anderes Bildteil

befindet, also im Wesentlichen in der Bildumrandung. Bei einigen Grafikmodi wird dieses Register auch als normale Bildschirmfarbe angesprochen (z.B. beim ANTIC-Modus "8" oder "A"). Für alle diese Register erzeugt das Betriebssystem Schattenregister.

Die Register und Schattenregister befinden sich an folgenden Speicherstellen im Atari-System:

Register			Schattenregister		
COLPF0	53270	$d016	COLPF0§	708	$2c4
COLPF1	53271	$d017	COLPF1§	709	$2c5
COLPF2	53272	$d018	COLPF2§	710	$2c6
COLPF3	53173	$d019	COLPF3§	711	$2c7
COLBAK	53174	$d01a	COLBAK§	712	$2c8

Wie schon am Anfang des Kapitels erwähnt, ist es möglich, den GTIA dazu zu bringen, die Bilddaten, die er vom ANTIC erhält, auf andere Art als normal darzustellen. Auch das BASIC enthält Grafikbetriebsarten ("Grafikgänge"), die nicht zu den sonst üblichen Darstellungen des ANTIC passen. Gemeint sind die Modi, die man durch "GRAPHICS 9", "GRAPHICS 10" oder "GRAPHICS 11" erhält.

Ursprünglich hat Atari seine Anlagen mit einem Chip namens "CTIA" ausgeliefert. Dieser Chip ist besonders für das Amerikanische Farbsystem (NTSC-Norm) konzipiert. Als klar wurde, daß Atari in Europa nicht nur ein paar Rechner verkaufen würde, wurden die weiteren Geräte mit dem GTIA-Chip ausgerüstet. Der GTIA ist besonders für europäische Farbsysteme ausgelegt. Fast alle Atari-Systeme in Europa enthalten den GTIA (in jedem Fall alle neuen Geräte, aber auch die meisten Atari 400 und 800). Die neuesten Geräte in den USA enthalten inzwischen ebenfalls einen weiterentwickelten Chip anstelle des alten CTIA.

Zwischen GTIA und CTIA gibt es einen wesentlichen Unterschied. Bei der Grafik mit CTIA und ANTIC werden nur die Grafikregister, die auch mit dem "SETCOLOR"-Befehl erreicht werden können, verwendet. Ohne Player-Missile-Grafik bleiben also Farbregister unbenutzt. Der GTIA-Chip kann dagegen alle neun Farbregister auch ohne Benutzung der Player-Missile-Grafik ansprechen, ermöglicht dem Benutzer also eine bessere Ausnutzung des Atari-Systems.

Da bei den "neuen" Systemen nur statt des CTIAs der GTIA verwendet wurde, nicht jedoch auch ein anderer ANTIC, muß zwangsläufig der GTIA allein für die neuen Modi ("GRAPHICS 9, 10 oder 11)"zuständig" sein. Tatsächlich betreibt man den ANTIC in diesen Modi auch normal, zum Beispiel in einem Schriftmodus oder auch einem Modus für hochauflösende Grafik, der GTIA stellt die Informationen des ANTIC lediglich anders dar.

Der neue GTIA ist zum CTIA kompatibel, das heißt, daß alle Programme, die auf dem CTIA laufen, auch für den GTIA geeignet sind. Der umgekehrte "Weg" ist aber nur möglich, wenn nur CTIA-Möglichkeiten ausgenutzt worden sind.

Der GTIA interpretiert die Bilddaten des ANTIC anders als in der gewohnten Weise. Er faßt die Daten zu jeweils vier Bit breiten Nibbles zusammen. Diese Nibbles steuern dann, welche Farben auf den Bildschirm gelangen. Es muß allerdings beachtet werden, daß es bei Verwendung der horizontalen Feinverschiebung oft unerwünschte Ergebnisse gibt, wenn diese Möglichkeiten des GTIA ausgenutzt wedrden.

Die unterschiedlichen Betriebsarten werden über die Bits 6 und 7 des Registers GTIACNTL gewählt. Die Bits 0-5 des GTIACNTL werden für die Player-Missile-Grafik verwendet. Bei der Programmierung ist also darauf zu achten, daß es nicht zu Konflikten mit der Player-Missile-Grafik kommt. Auch für GTIACNTL gibt es ein Schattenregister (GTIACNTL§). Die Register befinden sich an folgenden Stellen:

GTIACNTL 53275 $d01b
GTIACNTL§ 623 $26f

Die möglichen Kombinationen von Bit 6 und Bit 7 führen zu folgenden Funktionen:

Bit7	Bit6	Beschreibung
0	0	Diese Bitkombination liegt beim CTIA immer vor. Wenn Programme von einem Atari mit GTIA auf einem Atari mit CTIA (der in Europa selten ist !) laufen sollen, muß diese Bitkombination vorliegen. Bei dieser Kombination arbeitet der Atari wie gewohnt.
0	1	Jetzt interpretiert der GTIA die erwähnten Nibbles als Helligkeitsstufen. Dabei nimmt er die Farbe, die in Register COLBAK bzw. COLBAK§ steht und als Helligkeit die Bitkombination des jeweiligen Nibbles. Wenn der ANTIC beispielsweise den ANTIC-Modus "F" (BASIC: "Graphics 8") benutzt, faßt der GTIA je 4 Bildpunkte zu einem eine Bildzeile hohen Streifen zusammen, der die Farbe aus COLBAK hat. Die Bitkombination der normalerweise erscheinenden vier Punkte bestimmt die Helligkeit des Streifens. Dieses Beispiel entspricht dem "BASIC-Gang 9".
1	0	Jetzt interpretiert der GTIA die Nibbles als Nummer eines Farbregisters. Da der GTIA aber nur über neun Farbregister verfügt, ist die Zahl der gleichzeitig darstellbaren Farben auf neun beschränkt. Wenn der ANTIC beispielsweise den Modus "F" benutzt, faßt der GTIA je vier Bildpunkte zu einem eine Bildzeile hohen Streifen zusammen. Das von dem jeweiligen Nibble (der Bitkombination der normalerweise erscheinenden vier Punkte) angewählte Farbregister bestimmt, aus welchem Farbregister Farbe und Helligkeit des Streifens stammen. Dieses Beispiel entspricht dem "BASIC-Gang 10".

1 1 Jetzt interpretiert der GTIA die Nibbles als Farbstufen. Dabei nimmt er die Helligkeit, die in Register COLBAK bzw. COLBAK§ steht. Die Farbe wird durch die Bitkombination des Nibbles gewählt. Dabei gilt die normale Farbtabelle. Wenn der ANTIC beispielsweise den Modus "F" benutzt, faßt der GTIA je vier Bildpunkte zu einem eine Bildzeile hohen Streifen zusammen. Dieser Streifen hat dann die Helligkeit aus Register COLBAK und die Farbe, die durch die Bitkombination, der normalerweise erscheinenden, vier Bildpunkte bestimmt wird. Dieses Beispiel entspricht dem "BASIC-Gang 11".

Durch diese Möglichkeiten des GTIA erhöht sich die Zahl der gleichzeitig auf dem Bildschirm darstellbaren Farben erheblich. Auch die Auflösung ist im Verhältnis zu der Anzahl der gleichzeitig darstellbaren Farben sehr hoch. Gerade Darstellungen für die Statistik oder Spiele lassen sich oft so aufbauen, daß es nicht weiter stört, daß der Bildschirm aus kurzen, einzeiligen Linien besteht.

Natürlich kann man praktisch jedes Bild, das vom ANTIC erzeugt wird, in der beschriebenen Art vom GTIA darstellen lassen. Gerade mit den Schriftmodi des ANTIC lassen sich sehr farbreiche Bilder erzeugen, die im Speicher nur wenig Raum verbrauchen, sich daher also schnell von der CPU verändern lassen. Dazu ist es aber notwendig, sich einen eigenen Zeichensatz zu definieren.

Eine weitere Möglichkeit, interessante Effekte zu erzielen und mehr Farben als üblich auf den Bildschirm zu bringen, ist der Wechsel von Farbregisterinhalten während einer ANTIC-Programm-Unterbrechung. Während der Unterbrechung trägt man in die Farbregister neue Werte ein. Auf diese Art kann man mehr Farben gleichzeitig auf dem Bildschirm darstellen, als Farbregister vorhanden sind. Auch die Betriebsart der GTIA läßt sich während einer ANTIC-Programm-Unterbrechung variieren.

Mit dem Basic-Befehl "SETCOLOR R,F,H" schreibt man in die Schattenfarbregister bestimmte Werte, die die Farbe und Helligkeit codiert enthalten (die "Codierung" wurde bereits beschrieben). Für die möglichen "R" im Befehl wird jeweils ein Schattenfarbregister angesprochen:

für R = 0 : COLPF0§
für R = 1 : COLPF1§
für R = 2 : COLPF2§
für R = 3 : COLPF3§
für R = 4 : COLBAK§

Statt "SETCOLOR R,F,H" läßt sich auch

" POKE (Adresse des Schattenfarbregisters), F * 16 + H "

eingeben. Mit diesem Befehl ist auch die Wahl der Farbe der Farbregister für die Player-Missile-Grafik möglich, dies läßt sich mit dem "SETCOLOR"-Befehl nicht machen.

```
**********************************
*                                *
*                                *
*  DIE PLAYER-MISSILE-GRAFIK     *
*  -------------------------     *
*                                *
**********************************
```

Zu den wichtigsten Elementen bei Spielen gehören natürlich bewegliche Objekte. Bei herkömmlichen Systemen ist der Bildschirm ein einfacher Bereich des Speichers, der auf eine bestimmte Art auf den Monitor "projiziert" wird.

Ein auf dem Bildschirm zusammenhängendes Feld, das einen Spieler oder ein Geschoß darstellen soll, liegt im Speicher meist nicht direkt hintereinander. Zwischen den Informationen, die zu einem Spieler gehören, befinden sich Bytes, die zu den Bildteilen, die von einem Teil des Spielers in einer Zeile bis zum gleichen Spieler in der nächsten Zeile folgen, gehören. Dies muß bei jeder Operation mit einem Spieler beachtet werden. Die CPU muß also, wenn sie zum Beispiel das Aussehen eines Spielers ändern will, nur jedes soundsovielte Byte ändern. Dies bringt einen erheblichen Rechenaufwand mit sich.

Womöglich liegt ein Spieler auch noch halb in einem oder mehreren anderen Bytes, dann muß die CPU auch noch einzelne, zum Spieler gehörige und nicht zum Spieler gehörige Bits trennen muß. Dies bringt weiteren Rechenaufwand.

Zudem überdeckt jeder Spieler Bildteile oder wird von Bildteilen überdeckt. Jedes überdeckte Bildteil oder jeder überdeckte Spieler müssen nach dem Weiterbewegen natürlich wiederhergestellt werden. Also muß bei jedem Bildüberschreiben für einen Spieler zwischengespeichert werden, was gerade verdeckt wird.

Der gesamte Rechenaufwand läßt sich dadurch verringern, daß man die Spieler immer mehrere Punkte weit bewegt. Allerdings wirkt die Bewegung dann nicht mehr flüssig, sondern ruckend. Eine Lösung bringt die Einführung von Spielern und Geschossen, die nicht mehr per Software, sondern per Hardware eingeblendet werden. Auch Trefferkontrollen lassen sich hardwaremäßig leichter durchführen als per Software.

Bei den Atari-Geräten übernimmt dies die sogenannte Player-Missile-Grafik. Wenn auch der Aufwand nicht auf Null absinkt und der Atari sicher auch nicht das einfachste Konzept für bewegliche Objekte hat, sollte man sich immer vor Augen halten, welche Operationen der CPU man einspart. Außerdem ist die Player-Missile-Grafik von Atari extrem flexibel.

Mit der Player-Missile-Grafik ist es möglich, bis zu vier

Spieler und vier Geschosse in jedes beliebige Bild des ANTIC einzublenden. Diese Zahl läßt sich jedoch erhöhen, man kann auch mehrere Objekte mit einem Spieler oder einem Geschoß darstellen. Innerhalb von einer Zeile lassen sich jedoch nie mehr als vier Geschosse und vier Spieler darstellen.

Wenn zwei Objekte gemeinsam einen Spieler oder ein Geschoß "benutzen" sollen, können entweder ohne DMA, je nachdem welche Zeile des Bildes gerade aufgebaut wird, die Daten des einen oder des anderen Objekts in das Grafikregister geschrieben werden. Es ist aber auch mit DMA möglich, zwei Objekte durch einen Spieler darzustellen. Man muß lediglich in einer Bildzeile eine ANTIC-Programm-Unterbrechung auslösen und in der Unterbrechungsroutine das Register PMBASE (das Register ist im ANTIC-Kapitel beschrieben) auf die Basisadresse eines anderen Speicherfeldes legen.

Die Player-Missile-Grafik wird über das Register PMCNTL ein- und ausgeschaltet. Das Register PMCNTL befindet sich bei Adresse 53277 bzw. $d01d. Die einzelnen Bits des Registers haben folgende Funktionen:

Bit 0: Wenn dieses Bit auf "1" steht, ist die Übertragung der Grafikregister der Geschosse auf den Bildschirm eingeschaltet. Somit werden die Geschosse mit diesem Bit eingeschaltet.

Bit 1: Wenn dieses Bit auf "1" ist, ist die Übertragung der Grafikregister der Spieler auf den Bildschirm eingeschaltet. Somit werden die Spieler mit diesem Bit eingeschaltet.

Bit 2: Wenn dieses Bit auf "1" ist, können die Triggereingänge nur auf "0" gesetzt werden, das Loslassen einer Joystick-Feuertaste wird dadurch nicht mehr registriert. Damit ist das Speichern eines Tastendruckes auf eine der "Joystick-Feuertasten" möglich.

Die Farben der Spieler werden wie die Farben des Spielfeldes in Farbregistern gespeichert. Dabei haben immer ein Spieler und ein Geschoß gemeinsam ein Farbregister. Die Register sind in ihrer Funktion mit den anderen Farbregistern identisch. Die Register befinden sich an folgenden Adressen im Atari:

COLPM0	53266	$d012	Spieler/Geschoß 0
COLPM0§	704	$2c0	(Schattenregister)
COLPM1	53267	$d013	Spieler/Geschoß 1
COLPM1§	705	$2c1	(Schattenregister)
COLPM2	53268	$d014	Spieler/Geschoß 2
COLPM2§	706	$2c2	(Schattenregister)
COLPM3	53269	$d015	Spieler/Geschoß 3
COLPM3§	707	$2c3	(Schattenregister)

Die Form der Spieler wird in sogenannten Spieler- und Geschoßgrafikregistern abgelegt. Der GTIA unterstützt aber nur jeweils eine Bildzeile. Das heißt, daß nach jeder Zeile, sofern sich das Bild des Spielers zu dieser Zeile hin ändern soll, neue Daten in die Spieler- und Geschoßgrafikregister gebracht werden müssen. Dies läßt sich sowohl aus einem Assemblerprogramm heraus ausführen, als auch durch die ANTIC-DMA erledigen. Über die Benutzung der ANTIC-DMA gibt das Kapitel über den ANTIC Auskunft.

Für jeden Spieler ist ein Grafikregister vorhanden:

GRAFP0	53261	$d00d
GRAFP1	53262	$d00e
GRAFP2	53263	$d00f
GRAFP3	53264	$d010

Für die Geschosse ist ein gemeinsames Grafikregister vorhanden, in das die Daten aller Geschosse geschrieben werden:

GRAFPM 53265 $d011

Zu jedem Geschoss gehören zwei Bits in diesem Register:

Bit 0-1 Geschoß 1
Bit 2-3 Geschoß 2
Bit 4-5 Geschoß 3
Bit 6-7 Geschoß 4

Die Programmierung des Geschoßregisters ist etwas komplizierter als die Programmierung der Spielergrafikregister, da hier die einzelnen Bits der Geschosse zu einem Byte zusammengefügt werden müssen. Auch die DMA erleichtert diese Arbeit nicht. Aber trotzdem lassen sich Spiele auch mit dieser Form der Player-Missile-Grafik leichter programmieren, als völlig ohne Hardwarehilfe für bewegliche Objekte.

Die horizontale Position der Geschosse und Spieler wird ebenfalls in Register geschrieben. Der Registerinhalt besagt jeweils, beim wievielten Farbtakt einer Bildzeile angefangen wird, den Inhalt des Grafikregister (also das Objekt) auf den Bildschirm zu bringen.

Zu beachten ist aber, daß der linke Spielfeldrand je nach der Anzahl der Farbtakte pro Bildzeile, die man im ANTIC einstellen kann, bei verschiedenen Farbtakten liegt. Die Farbtakte vor dem eigentlichen Bild gehören zum Bildrand, der die Farbe aus COLBAK hat. Die ersten Farbtakte liegen normalerweise außerhalb des Fernseherbildes. Um Objekte unsichtbar zu machen, kann man sie ohne weiteres in diesen Bildbereich bewegen, man muß das Horizontalregister des Spielers nur auf Null setzen.

Für jeden Spieler und jedes Geschoß ist ein Horizontalregister vorhanden:

HPOSP0	53248	$d000	(Spieler 0)
HPOSP1	53249	$d001	(Spieler 1)
HPOSP2	53250	$d002	(Spieler 2)
HPOSP3	53251	$d003	(Spieler 3)
HPOSM0	53252	$d004	(Geschoß 0)
HPOSM1	53253	$d005	(Geschoß 1)
HPOSM2	53254	$d006	(Geschoß 2)
HPOSM3	53255	$d007	(Geschoß 3)

Außerdem gibt es ein weiteres Register, um die vertikale Position von Spielern und Geschossen zu beeinflussen:

VDELAY 53276 $d01c

VDELAY wird benutzt, wenn man zweizeilige Auflösung gewählt hat, aber ein Objekt trotzdem in eine bestimmte Zeile bringen möchte. Durch eine "1" in einem Bit wird bewirkt, daß das entsprechende Objekt um eine Zeile nach unten gerückt wird. Das Register ist folgendermaßen belegt:

Bit 0	Geschoß 0
Bit 1	Geschoß 1
Bit 2	Geschoß 2
Bit 3	Geschoß 3
Bit 4	Spieler 0
Bit 5	Spieler 1
Bit 6	Spieler 2
Bit 7	Spieler 3

Die horizontale Größe der Objekte wird ebenfalls in Registern gespeichert. Die horizontale Größe ist die Anzahl der Farbtakte, die ein Objekt in einer Zeile einnimmt. Für jeden Spieler gibt es ein derartiges Register:

SIZEP0	53256	$d008	(Spieler 0)
SIZEP1	53257	$d009	(Spieler 1)
SIZEP2	53258	$d00a	(Spieler 2)
SIZEP3	53259	$d00b	(Spieler 3)

Die horizontale Größe wird durch Bit 0 und Bit 1 der Register bestimmt (die anderen Bits sind unbenutzt):

Bit1	Bit0	Größe
0	0	Normalgröße, jedes Bit ist einen Farbtakt breit (ein Spieler ist also 8 Farbtakte breit).
0	1	Doppelte Größe, jedes Bit ist zwei Farbtakte breit (ein Spieler ist also 16 Farbtakte breit).
1	0	Normalgröße, jedes Bit ist einen Farbtakt breit (ein Spieler ist also 8 Farbtakte breit).
1	1	Vierfache Größe, jedes Bit ist vier Farbtakte breit (ein Spieler ist also 32 Farbtakte breit).

Die Größen der Geschosse werden dagegen in dem Register

SIZEM 53260 $d00c

gespeichert.

Je zwei Bits des Registers bestimmen die Größe eines Geschosses:

Bit 0-1	Geschoß 0
Bit 2-3	Geschoß 1
Bit 4-5	Geschoß 2
Bit 6-7	Geschoß 3

Die möglichen Bitkombinationen bewirken folgendes:

0	0	Normalgröße, jedes Bit ist einen Farbtakt breit (das Geschoß ist also zwei Farbtakte breit).
0	1	Doppelte Größe, jedes Bit ist zwei Farbtakte breit (das Geschoß ist also vier Farbtakte breit).
1	0	Normalgröße, jedes Bit ist einen Farbtakt breit (das Geschoß ist also zwei Farbtakte breit).
1	1	Vierfache Größe, jedes Bit ist vier Farbtakte breit (das Geschoß ist also acht Farbtakte breit).

Wenn Teile des Bildschirms die gleiche Bildschirmposition haben wie Spieler oder Geschosse, stellt sich die Frage, was im Vordergrund und was im Hintergrund dargestellt wird. Dies läßt sich durch das Register GTIACNTL (bzw. GTIACNTL§) bestimmen. Dieses Register ist aber auch noch für weitere Fuktionen zuständig. So bestimmen Bit 6 und 7 die GTIA-Betriebsart. Die ersten 4 Bits dienen der Prioritätskontrolle. Durch Setzen eines der 4 Bits wählt man eine Prioritätsfolge zwischen den Spielfeldfarben und den Spielern und Geschossen:

(Die Priorität FÄLLT von links nach rechts !)

Bit 0:
  P0, P1, P2, P3, PF0, PF1, PF2, PF3/P5, BAK

Bit 1:
  P0, P1, PF0, PF1, PF2, PF3/P5, P2, P3, BAK

Bit 2:
  PF0, PF1, PF2, PF3/P5, P0, P1, P2, P3, BAK

Bit 3:
  PF0, PF1, P0, P1, P2, P3, PF2, PF3/P5, BAK

Wenn mehr als eins der 4 Bits gesetzt ist, so sind die Prioritäten nicht eindeutig festgelegt. Es kommt zu Prioritätskonflikten zwischen den einzelnen Bildobjekten. Die "Überlappungszone" zwischen derartigen, in einem Prioritätskonflikt stehenden Objekten erscheint schwarz. Wenn man zum Beispiel Bit 0 und Bit 2 von GTIACNTL setzt, so wird P0 schwarz erscheinen, wenn er über PF1 gerät. Das gleiche geschieht auch mit P1, P2 und P3 wenn sie über PF1, PF2 und PF3/P5 geraten.

In den ANTIC-Modi mit einer Farbe und 40 Zeichen pro Zeile wird die Leuchtstärke eines Zeichens ohne Rücksicht auf die Priorität von PF1 bestimmt. Wenn ein Spieler oder ein Geschoß mit höherer Priorität ein derartiges Zeichen überdeckt, so wird die Farbe dieser Zone von der Spielerfarbe bestimmt.

Die beiden verbleibenden Bits von GTIACNTL haben folgende Funktionen:

Bit 4: Wenn dieses Bit gesetzt ist, haben alle Geschosse die Farbe, die in COLPF3 steht (Farbe von PF3). Dadurch können die Geschosse auch zu einem fünften Spieler (P5) zusammengefaßt werden.

Bit 5: Das Setzen dieses Bits ermöglicht es, auch mehrfarbige Spieler abzubilden. Dabei werden Spieler 0 und Spieler 1 sowie Spieler 2 und Spieler 3, sofern sie sich überlappen, miteinander verknüpft. Die Priorität gilt dann nur noch bedingt. Es können die Farben beider Spieler erscheinen. Je nachdem, welches Bit welches Spielers gesetzt ist, erscheint eine von drei Farben (zwei Farben der Spieler und die Farbe unter den Spielern) auf dem Bildschirm .

Besonders bei Spielen ist es interessant, festzustellen ob es "Zusammenstöße" zwischen Spielern und Geschossen mit anderen Spielern, Geschossen und Bildteilen gab. Der GTIA besitzt insgesamt 16 Register, die ein Programm abfragen kann, um derartige Kollisionen zu erkennen.

Für jedes Geschoß gibt es ein Register, das Kollisionen zwischen dem jeweiligen Geschoß und den Spielfeldfarben speichert:

KOLMOPF	53248	$d000
KOLM1PF	53249	$d001
KOLM2PF	53250	$d002
KOLP3PF	53251	$d003

Ein Bit dieser Register signalisiert dadurch, daß es auf "1" steht, daß es eine Kollision zwischen dem Geschoß, dem das Register zugeordnet ist und der jeweiligen Spielfeldfarbe gab:

Bit 0 : Kollision mit PF0
Bit 1 : Kollision mit PF1
Bit 2 : Kollision mit PF2
Bit 3 : Kollision mit PF3
Bit 4-7 : unbenutzt (stehen auf "0")

Es gibt auch für jeden Spieler ein Register, das einen Zusammenstoß zwischen dem jeweiligen Spieler und einer Spielfeldfarbe signalisiert:

KOLP0PF	53252	$d004
KOLP1PF	53253	$d005
KOLP2PF	53254	$d006
KOLP3PF	53255	$d007

Ein Bit dieser Register signalisiert daurch, daß es auf "1" steht, daß eine Kollision zwischen dem Spieler, dem das Register zugeordnet ist und der jeweiligen Bildschirmfarbe stattgefunden hat:

Bit 0 : Kollision mit PF0
Bit 1 : Kollision mit PF1
Bit 2 : Kollision mit PF2
Bit 3 : Kollision mit PF3
Bit 4-7 : unbenutzt (stehen auf "0")

Desweiteren gibt es für jedes Geschoß ein Register, das signalisiert, ob es Kollisionen des jeweiligen Geschosses mit einem Spieler gab:

KOLM0P	53256	$d008
KOLM1P	53257	$d009
KOLM2P	53258	$d00a
KOLM3P	53259	$d00b

Ein Bit dieser Register signalisiert dadurch, daß es auf "1" steht, daß eine Kollision des Geschosses, dem das jeweilige Register zugeordnet ist, mit dem jeweiligen Spieler stattgefunden hat:

Bit 0 :	Kollision mit Spieler 0 (P0)
Bit 1 :	Kollision mit Spieler 1 (P1)
Bit 2 :	Kollision mit Spieler 2 (P2)
Bit 3 :	Kollision mit Spieler 3 (P3)
Bit 4-7 :	unbenutzt (stehen auf "0")

"Last but not least" gibt es für jeden Spieler ein Register, dessen Bits Zusammenstöße mit anderen Spielern signalisieren:

KOLP0P	53260	$d00c
KOLP1P	53261	$d00d
KOLP2P	53262	$d00e
KOLP3P	53263	$d00f

Ein Bit dieser Register signalisiert, daß ein Zusammenstoß zwischen dem Spieler, dem das Register zugeordnet ist und einem anderen Spieler stattgefunden hat. Das "eigene" Bit im jeweiligen Register steht immer auf "0".

Bit 0 :	Kollision mit Spieler 0 (P0)
Bit 1 :	Kollision mit Spieler 1 (P1)
Bit 2 :	Kollision mit Spieler 2 (P2)
Bit 3 :	Kollision mit Spieler 3 (P3)
Bit 4-7 :	unbenutzt (stehen immer auf "0")

Eine Kollision wird in dem Moment markiert, in dem die Position, an der die Kollision stattfindet, auf dem Bildschirm abgebildet wird. Direkt nach dem Einschreiben in die Positionsregister hat zwar genaugenommen die Kollision schon

stattgefunden, durch die interne Struktur des GTIA ist es aber zu diesem Zeitpunkt noch nicht möglich, die Kollision zu bemerken.

Die Bits in den Kollisionsregistern bleiben so lange gesetzt, bis in ein "Löschregister" irgendein Wert geschrieben wird. Die einzelnen Bits dieses Registers haben keine weitere Funktion. Es ist also wirklich völlig egal, welcher Wert in das Register

HITCLR 53278 $d01e

geschrieben wird.

Die Kollision ist also unter Umständen auch noch in dem Moment gespeichert, in dem die Objekte sich bereits wieder an anderen Stellen befinden. Auf diese Art kann man die Bewegungen der Objekte auch in Unterbrechungsroutinen ausführen, die Abfrage aber erst später im Hauptprogramm machen.

Wie benutzt man die Player-Missile-Grafik nun konkret ?

Zuerst muß man sich entscheiden, ob man die Objekte durch den DMA des ANTICs darstellen läßt oder direkt aus einem Assemblerprogramm in die Grafikregister des GTIA schreibt. Gewöhnlich wird man den DMA benutzen, da man ansonsten sehr genau darauf achten muß, welche Zeile gerade abgebildet wird. Der Aufand, der entsteht, wenn man ohne DMA mit der Player-Missile-Grafik arbeitet, ist nur sehr selten zu rechtfertigen.

Sofern man die Player-Missile-Grafik mit dem DMA benutzt, muß man sich einen freien Speicherbereich im Atari suchen. Gewöhnlich liegt ganz "oben" im freien Arbeitsspeicher der Bildschirmspeicher. Direkt darunter wird man gewöhnlich das ANTIC-Programm ansiedeln. Es empfiehlt sich, unter dem ANTIC-Programm das Speicherfeld für den Player-Missile-DMA unterzubringen. Falls man weitere Speicherfelder für die Player-Missile-Grafik anlegt, sollte der Speicher weiter von oben nach unten gefüllt werden. Die Adresse des Speicherfeldes für den Player-Missile-DMA muß dem ANTIC über das Register PMBASE mitgeteilt werden. Bei der Wahl der Speicherbereiche der Player-Missile-DMA ist jedoch zu beachten, daß, wie schon erwähnt, immer an einer 1K-Byte bzw. 2K-Byte Grenze beginnen müssen.

Die horizontale Spielerbewegung läßt sich direkt durch ein Verändern der Positionsregister im GTIA erreichen. Um eine vertikale Spielerbewegung zu bewirken, muß man die Bytes des Spielers oder die einzelnen Bits des Geschosses durch das Speicherfeld für den Player-Missile-DMA bewegen. Dafür sollte man sich jeweils Unterprogramme schaffen. Die Unterprogramme dafür sind jedoch relativ einfach, da der Speicherbereich für einen Spieler oder ein Geschoß nie mehr als 256 Bytes hat, also nie größer ist, als der größte mit einem Prozessorregister (Register A,X oder Y) darstellbare Wert. Die 16 Bit Operationen, die ohne Player-Missile-Grafik nötig wären, entfallen also weitestgehend.

Jetzt sollte der Speicherbereich für die Player-Missile-Grafik gelöscht werden. Außerdem müssen die Horizontalpositionen und Größen der Spieler, sowie der Grad der vertikalen Auflösung im ANTIC bzw. GTIA festgelegt werden. Die Spielerfarben werden in die Register COLPM0§ bis COLPM3§ (Schattenregister von COLPM0 bis COLPM3) eingetragen.

Ganz zuletzt sollte man den Player-Missile-DMA (mit Register DMACNTL) und die Player-Missile-Grafik (mit Register PMCNTL) einschalten; so treten keine Störungen auf dem Bildschirm während der Initialisierung auf.

```
***************************************************
*                                                 *
*                                                 *
*   DIE TRIGGEREINGÄNGE UND DIE KONSOLENTASTER    *
*   ------------------------------------------    *
*                                                 *
***************************************************
```

Die Triggereingänge des GTIA sind bei den alten Atari-Modellen mit den vier Joystickeingängen verbunden. Über die Triggereingänge fragt man so zum Beispiel die "Feuertasten" der Joysticks ab.

Da die neuen Atari-Modelle nur noch zwei Joystick-Ports besitzen, wurde einer der freien Triggereingänge zur Abfrage der Freigabe für ein ROM-Modul verwendet. Der vierte Triggereingang ist unbenutzt.

Für jeden Triggereingang steht ein Register zur Verfügung, das jeweils im niedrigsten Bit (Bit 0) den Status des Eingangs angiebt. Eine "1" in einem dieser Bits zeigt zum Beispiel einen nicht gedrückten Taster an den Joysticks an, eine "0" einen gedrückten Taster. Wie schon erwähnt, läßt sich das Zurückschalten auf "1" durch das Setzen von Bit 2 des Registers PMCNTL unterdrücken. Wenn eine "0" in einem der Triggereingangsregister steht, bleibt sie gespeichert, bis das Bit 2 von PMCNTL wieder gelöscht wird. Die Bits 1 bis 7 der Register stehen immer auf "0". Für jedes der Register steht ein Schattenregister zur Verfügung.

Die Register und Schattenregister befinden sich an folgenden Adressen:

TRIG0	53264	$d010	JOYSTICK 1
TRIG0§	644	$284	
TRIG1	53265	$d011	JOYSTICK 2
TRIG1§	645	$285	

TRIG2	53266	$d012	Freigabe Cartridge ROM
TRIG2§	646	$286	(Joystick 3)
TRIG3	53267	$d013	unbenutzt
TRIG3§	647	$287	(Joystick 4)

(in Klammern: Atari 400/800)

Die Konsolentaster "START", "SELECT" und "OPTION" werden ebenfalls über den GTIA abgefragt. Dafür steht das Register CONSOL zur Verfügung:

CONSOL 53279 $d01f

Die Konsolentaster sind den Bits des Registers folgendermaßen zugeordnet:

Bit 0 : START
Bit 1 : SELECT
Bit 2 : OPTION

Eine "0" in einem der Bits signalisiert, daß die betreffende Taste gedrückt ist. Durch Einschreiben einer "1" in ein oder mehrere Bits, wird die Eingabe durch den oder die betreffenden Taster gesperrt.

Bit 3 des CONSOL-Registers steuert das "Klicken" der Tastatur. Das Betriebssystem schreibt während des Vertikal-Leertaktes eine 1 in dieses Bit. Durch das Einschreiben einer "0" in dieses Bit löst man das "Klicken" aus. Das Bit sollte aber nur kurzzeitig auf "0" stehengelassen werden, insbesondere wenn man die Vertikal-Leertakt-Unterbrechungsroutine des Betriebssystems abschaltet.

Vor dem Lesen der Taster sollte man $08 in das Register CONSOL schreiben, um sicherzugehen, daß kein Taster gesperrt ist.

Das Aussehen der Farben ist bei NTSC- und PAL-System leider nicht identisch. Wenn man auf einem PAL-System ein Spiel schreibt, kann es unter Umständen nötig sein, die Farben zu ändern, bevor man das Programm auf einem NTSC-System laufen läßt. Die Änderungen können aber auch gleich im Programm berücksichtigt werden. Im Register

NTSCPAL 53268 $d014

wird dann im Programmablauf abgefragt, ob ein PAL- oder ein NTSC-Gerät vorliegt. Bei einem PAL-Gerät sind die Bits 1-3 "0", bei einem NTSC-Gerät "1".

Außerdem ist beim NTSC-System normalerweise nicht die gleiche Anzahl von Bildzeilen auf dem Bildschirm vorhanden, wie beim PAL-System.

DER POKEY

1) ALLGEMEINES

2) TONERZEUGUNG

3) TASTATURABFRAGE

4) PADDLESTEUERUNG

5) SERIELLE EIN-/AUSGABE

Der POTentiometer and KEYboard-Controller-Chip ist der dritte Spezialbaustein im Atari 600XL/800XL. Er übernimmt zum Einen fast die gesamte Tonsteuerung, zweitens ist er für die Kommunikation mit extern anzuschließenden Geräten zuständig und als letztes übernimmt er das Einlesen der Paddle-Kanäle.

Als Verbindung zur CPU stehen ihm die 8 Datenleitungen, Takt- Schreib-/Leseleitung, eine Interrupt- und vier Adreßleitungen zur Verfügung. Über eine Hardware-Resetleitung verfügt der POKEY nicht.

Mit seinen vier Adreßleitungen können im POKEY 16 verschiedene Daten- beziehungsweise Steuer-Register angesprochen werden.

Diese 16 Register teilen sich hauptsächlich in 9 Tonregister, 4 Übertragungsregister, ein nicht belegtes und ein Interruptverarbeitungsregister auf.

```
********************
*                  *
*                  *
*  TONERZEUGUNG    *
*  ------------    *
*                  *
********************
```

Der POKEY besitzt die Möglichkeit, maximal vier Tonkanäle gleichzeitig oder in Kombination miteinander zu verwalten.

Jeder Ton, den wir hören können, setzt sich aus mehreren Komponenten zusammen. Da ist zum Einen die Tonhöhe, die technisch als Frequenz bezeichnet wird und in Hz (Hertz, nach dem bekannten Physiker Heinrich Hertz) gemessen wird.

Ein Hz ist nun eine komplette Schwingung pro Sekunde, wobei 'komplett' bedeutet, daß bei immer wiederkehrenden Schwingungsbildern man sich einen beliebigen Punkt der Schwingung sucht und dann so lange wartet, bis genau dieser Punkt wieder erreicht wird. Der Bereich zwischen den beiden gefundenen Punkten ist dann eine komplette Schwingung. Hat man nun 1000 Schwingungen pro Sekunde, so besitzt das Signal eine Frequenz von 1000Hz, oder auch 1kHz.

Je größer die Frequenz des Signals ist, umso höher empfinden wir den Ton, beziehungsweise das Geräusch.

Eine weiteres Charakteristikum einer Schwingung ist die Schwingungsform. Als Extrema wären hier z.B. die reine Sinus-Schwingung oder, als Gegenstück dazu, die reine Rechteck-Schwingung zu nennen. Dies sind jedoch Sonderfälle, die sich zwar heutzutage mit elektronischen Mitteln recht sicher und gut herstellen lassen, die jedoch wegen ihrer Ausgeprägtheit im Klang derart 'nervend' sind, daß sie sich kaum jemand länger anhören kann. Wer das nicht glaubt, sollte sich einmal den 440Hz-Normton (Kammerton a) am Telefon fünf Minuten lang anhören.

Alle Geräusche und Töne, die wir heutzutage um uns haben, sind Mischformen unterschiedlichster Signalformen. Aus der jeweiligen Signalzusammensetzung können wir, in Verbindung mit den sogenannten Hüllkurven, unterschiedliche Signalquellen recht sicher voneinander unterscheiden. So dürfte eine Verwechselung zwischen einem Waldhorn und einem Klavier einigermaßen selten vorkommen.

Die Hüllkurve beschreibt nun, wie sich die Lautstärke des jeweiligen Tones im Verhältnis zur Zeit verändert. Da gibt es z.B. Signale, die sehr schnell laut sind, um dann langsam und gleichmäßig abzuklingen (Klavier), die Violine dagegen erreicht wesentlich langsamer Ihr Lautstärkemaximum.

Als letzte wesentliche Eigenschaft jedes Tones verarbeiten wir mehr oder weniger unbewußt den Frequenzbereich, in dem sich das Signalgemisch ungefähr befindet. Es hört sich einfach anders an, wenn man Musik direkt aus der HiFi-Stereoan-

lage hört, als wenn man dies über das Telefon tut. Nicht unbedingt wegen eventueller Störungen und Knack-Geräusche, sondern wegen des enorm eingeschränkten übertragenen Frequenzbereichs. Der ist von der Post so bestimmt, daß die übertragenen Töne hauptsächlich die Tonhöhen durchlassen, die zur Übermittlung von Sprache wesentlich sind.

Im POKEY sind nun vier Register enthalten, die nichts weiter tun, als ein vorgegebenes Taktsignal mit bestimmter Frequenz durch einen bestimmten, ebenfalls vorzugebenden Wert zu teilen. Dadurch entsteht dann eben ein niedrigerer Ton. Da das jedoch zu primitiv wäre, und sich damit nach dem oben gesagten keine vernünftigen Klänge erzeugen ließen, gibt es nun noch weitere Möglichkeiten. Als erstes kann man z.B. zwei solcher Zähler aneinanderhängen, das heißt also, den Ausgang des einen Zählers mit dem Eingang des zweiten zu verbinden. So kann Zähler 2 an Zähler 1 angehängt werden, genauso wie Zähler 4 an Zähler 3. Dieses Anhängen hat folgenden Zweck:

Durch die Größe der einzelnen Zählregister von 8 Bit ergibt sich ein mögliches Teilerverhältnis im Bereich von 1 bis 256 (intern wird auf den Registerwert eine %1 addiert). Hängt man nun zwei Zähler aneinander, so erhält man ein mögliches Verhältnis von 1 bis 65536, da ja nun 16 Bit zur Verfügung stehen.

Es gibt somit die Möglichkeit, vier 8bit-Kanäle oder einen 16bit-Kanal und zwei 8bit-Kanäle oder, als letztes, zwei 16bit-Kanäle zu betreiben. Alles dies wird über entsprechende Satusbits festgelegt.

Die eigentlichen Teilerverhältniswerte werden in die entsprechenden Register AUDIFREQ1 bis AUDIFREQ4 geschrieben.

Als Takt kann für jedes der Register ein 64kHz-, 15kHz- oder, und das gilt nur für Kanal 1 und 3, ein 1,77MHz-Takt gewählt werden. Je höher der Takt gewählt wird, desto höher ist bei gleichem Teilerverhältnis das Ausgangssignal.

Aus der Kombination der Takteingangsfrequenzen und der Teilerverhältnisse lassen sich nun sämtliche Frequenzbereiche von 0 bis 1,77/2 MHz= 887kHz erreichen. Die letzte Teilung durch den Faktor zwei ist immer gegeben und hat den Zweck, immer nur symmetrische Signale auszugeben. Hat man ein Digitalsignal als letztes durch den Faktor zwei geteilt, so ist dieses Signal genauso lange 'an' wie 'aus', also zeitlich symmetrisch. Diese Bedingung ist wesentlich, denn ein unsymmetrisches Signal hört sich völlig anders an als ein symmetrisches, so daß sich bei Veränderung von Teilerwerten nicht nur die Frequenz, sondern auch die Signalform, und somit der Klang ändern würde, wenn nicht zum Schluß das Signal immer in ein symmetrisches abgeändert würde.

Neben der Frequenz läßt sich auch noch die Art und Form eventueller Verzerrung einstellen. Dazu wird das normale Ausgangssignal, das von den Zählregistern geliefert wird, logisch verknüpft mit zyklisch auftretenden 'Störungen'.

Diese Störungen werden über sogenannte Schieberegister erzeugt, die wie eine lange Röhre funktionieren. Schiebt man in das eine Ende der Röhre Signale hinein, kommt irgendwann einmal am anderen Ende eben dieses Signal heraus. Nimmt man dieses Signal dann wieder, um es am vorderen Ende wieder hineinzutun, hat man einen Zyklus, der endlos weitergeht. Um nun nicht einen völlig 'langweiligen', einmaligen Rhythmus herauszubekommen, wendet man noch einen weiteren Trick an: Die Röhre wird an einigen Stellen 'angebohrt', und die an der Stelle anliegenden Signale zusammen mit dem End-Signal auf den Röhreneingang gelegt. Dadurch ist immer noch sichergestellt, daß es sich bei dem ganzen Vorgang um einen zyklischen handelt, aber dessen Wiederholrate ist wesentlich geringer.

Es sind nun im POKEY drei dieser rückgekoppelten Schieberegister, die als Polynom-Zähler bezeichnet werden, vorhanden. Es gibt einen 4 Bit langen, einen mit 5 Bit und einen, der umschaltbar entweder 17 oder 9 Bit Länge aufweist.

Die Frequenz wurde, wie schon erwähnt, durch die Werte in den Frequenzregistern AUDIFREQ1 bis AUDIFREQ4 festgelegt.

Die Verzerrung und die 16 stufige Lautstärke werden hingegen über die Kontrollregister AUDICNTL1 bis AUDICNTL4 programmiert. Die einzelnen Bits dieser Kontrollregister bedeuten:

Bit	Bedeutung
0	Diese vier Bit sind
1	zuständig für die Lautstärke.
2	Es sind Werte zwischen $00 und
3	$0f möglich (%0000 bis %1111).
4	VOLUME_ONLY
5	Diese drei Bit werden zur
6	Auswahl einer der sechs Verzer-
7	rungsmöglichkeiten benutzt (s.Tab).

Tabelle für die Wertigkeiten der Bits 5,6 und 7 der Register AUDICNTLn:

Bit 7	6	5	Signalmischung
0	0	0	Das vom Zählregister stammende Signal wird zuerst mit dem 5bit-Polynom, dann mit dem 17bit-Polynom verknüpft, bevor es auf die letzte Teilerstufe durch den Faktor zwei geschickt wird.
0	x	1	Es findet nur eine Verknüpfung mit dem 5bit-Polynom statt vor der letzten Division.

Bit 7	6	5	Signalmischung (Fortsetzung)
0	1	0	Das vom Zählregister stammende Signal wird zuerst mit dem 5bit-Polynom, dann mit dem 4bit-Polynom verknüpft, bevor es auf die letzte Teilerstufe durch den Faktor zwei geschickt wird.
1	0	0	Die Ausgangsfrequenz wird mit dem Signal des 17bit-Polynoms verknüpft und dieses Signal auf die letzte Teilerstufe geschickt.
1	x	1	Diese 2 Kombinationen ergeben einen reinen Ton, da keinerlei Verknüpfung des Ausgangssignals mit einem der Polynome geschieht, sondern nur die letzte Justage-Division.
1	1	0	Es erfolgt nur eine Verknüpfung der Ausgangsfrequenz mit dem vier bit breiten Polynom. Danach wird das Ergebnis, genau wie alle anderen, noch einmal durch den Wert zwei geteilt.

Das 'x' in der Tabelle bedeutet, daß hier '0' oder '1' stehen darf.

Das VOLUME_ONLY-Bit aus jedem der vier AUDICNTLn-Register schaltet die gesamte interne Zähl-, Verzerrungs- und Verknüpfungsmaschinerie aus und bestimmt, daß als Ausgangssignal genau die der Lautstärke entsprechende Spannung ausgesendet werden soll. Ist also VOLUME_ONLY gesetzt und die Lautstärke auf 16 (volle Lautstärke), so wird die im Fernseher/Monitor befindliche Lautsprechermembran so weit wie möglich aus ihrer Ruheposition herausgelenkt. Setzt man dann die Lautstärke wieder auf Null zurück, so fällt die Membran ebenfalls wieder in ihre Ruheposition zurück, es ergibt sich ein Knacken im Lautsprecher.

Die Aufgabe dieser Zusatzeinrichtung ist natürlich nicht, nur ein Knackgeräusch zu erzeugen. Mit dieser Methode unterliegt der Programmierer annähernd keiner Beschränkung bezüglich Signalform und Frequenz, da jede einzelne Membranbewegung des Lautsprechers programmiert werden kann. Es ist jedoch bei allzu schnellen Amplitudenänderungen (Amplitude= Höhe des Ausschlags bei einer Schwingung) darauf zu achten, daß zwar der Atari diese Signale herausgeben kann, harte Kurven jedoch durch die dann folgende Modulation/Demodulation im Fernseher erheblich abgerundet werden. Dies dürfte jedoch im Normalfall keine Einschränkung darstellen, da ein Fernseher heutzutage oft schon HiFi-Qualitäten besitzt.

Im Großen und Ganzen jedoch kann gesagt werden, daß zumindest mit entsprechendem Softwareaufwand der Atari ohne Zusatzgeräte alle gewünschten Akustikeffekte erzeugen kann.

Man denke da z.B. an Softwarepakete, die schon die alten 400/800 ohne zusätzliche Hardware haben sprechen lassen. Die Sprache war sogar mittelmäßig verständlich.

Neben diesen vier, auf jeden Kanal einzeln bezogenen Registern existiert noch ein weiteres, für die Tonerzeugung wesentliches Register: AUDIOCOM.

Diese Adresse beinhaltet die folgenden 8 Steuerbits:

BIT	Erläuterung
0	Schaltet den Haupttakt für die vier Frequenzregister von 64kHz auf 15kHz zurück, wenn es gesetzt wird.
1	Ist dieses Bit gesetzt, wird in den Ausgang von Kanal 2 ein Hochtonfilter eingesetzt, dessen Characteristik von Kanal 4 bestimmt wird.

BIT	Erläuterung (Fortsetzung)
2	In den Ausgang von Kanal 1 wird ein von Kanal 2 gesteuertes Hochtonfilter eingefügt.
3	Durch Setzen dieses Bits wird Kanal 4 an Kanal 3 angehängt, so daß ein 16bit-Register entsteht.
4	Ist dieses Bit gesetzt, wird Kanal 2 an Kanal 1 gehängt.
5	Dieses Bit bewirkt, daß Kanal 3 mit einer Grundfrequenz von ca. 1,77 MHz betrieben wird.
6	Das gleiche wie bei Bit 5 für Kanal 1.
7	Beim Setzen dieses Bits wird das 17bit-Polynom in ein 9bit langes Polynom verwandelt.

Als allgemeine Berechnungsgrundlage für das Verhältnis von Ein- zu Ausgangsfrequenz kann die Gleichung

Ausgangsfrequenz = Eingangsfrequenz / 2 (AUDIFREQn + Off)

verwendet werden, wobei Off den Wert 4 erhält, wenn es sich bei dem Teiler um einen 8bit-Teiler handelt. Ist es dagegen ein 16bit-Teiler, muß Off aus technischen Gründen mit 7 angesetzt werden.

Neben den Tonerzeugungsaufgaben besitzen diese Zähler noch eine weitere Aufgabe.

Wenn einer der Zähler 1, 2 oder 4 auf Null zurückgezählt hat, wird automatisch vom POKEY ein Interrupt ausgelöst. Die Zähler können also genauso gut als Timerbausteine fungieren.

Um alle Frequenzregister auf die Werte zu setzen, die Programmiert wurden (vor allem sinnvoll beim Betrieb als Timer!), wird einfach irgend ein Wert in Adresse STIMER geschrieben.

Als 'Abfallprodukt' der Polynomzähler entsteht in Register RANDOM eine 8bit-Zufallszahl, die dort entsprechend ausgelesen werden kann. Sie besteht aus den höchsten 8 Bit des 17-/9-bit Polynomzählers.

Abschließend zu diesem Thema folgt noch eine Tabelle, aus der die Teilerverhältnisse der Frequenzregister bei bestimmten Notenwerten hervorgehen. Dazu sind folgende Werte vorauszusetzen:

AUDIOCOM = $00 ; also nichts 'Besonderes'

AUDICNTLn = $Ax ; x ist die Lautstärke von 0..15

Mit den genannten Daten, die nur ein Beispiel sein sollen, ergeben sich die auf der umseitig gedruckten Tabelle folgende Werte für AUDIOFREQn:

Es lassen sich aber auch praktisch alle anderen Frequenzen und somit Notenwerte mit dem Atari erreichen. Dies wird auch in Vergleichen zwischen den einzelnen Mikrocomputern oft falsch dargestellt. Der Atari verfügt nicht nur über 3 1/2 Oktaven Frequenzumfang. Die höchsten erzeugbaren Frequenzen liegen weit über der höchsten wahrnehmbaren Frequenz. Auch die tiefste erzeugbare Frequenz ist nicht mehr hörbar.

Note	AUDIFREQn Hexadezimal	Dezimal
C (tiefes)	$f3	243
Cis oder Des	$e6	230
D	$d9	217
Dis oder Es	$cc	204
E	$c1	193
F	$b6	182
Fis oder Ges	$ad	173
G	$a2	162
Gis oder As	$99	153
A	$90	144
Ais oder B	$88	136
H	$80	128
c (mittleres)	$79	121
cis oder des	$72	114
d	$6c	108
dis oder es	$66	102
e	$60	96
f	$5b	91
fis oder ges	$55	85
g	$51	81
gis oder as	$4c	76
a	$48	72
ais oder b	$44	68
h	$40	64
c'	$3c	60
cis' oder des'	$39	57
d'	$35	53
dis' oder es'	$32	50
e'	$2f	47
f'	$2d	45
fis' oder ges'	$2a	42
g'	$28	40
gis' oder as'	$25	37
a'	$23	35
ais' oder b'	$21	33
h'	$1f	31
c''	$1d	29

```
************************
*                      *
*                      *
*   TASTATURABFRAGE    *
*   ---------------    *
*                      *
************************
```

Wird irgend eine Taste auf der Tastatur des 600XL/800XL gedrückt, die nicht eine der seitlich montierten Spezialtasten oder SHIFT oder CONTROL ist, wird ein Interrupt vom POKEY ausgelöst, der dem Betriebssystem Mitteilung von dem Tastendruck machen soll. Von der entsprechenden Interruptroutine wird dann der Tastaturkode aus Register KBCODE ausgelesen und für die weitere Verarbeitung im Vertical Blank Interrupt gesichert.

Eine Ausnahme hiervon stellt die BREAK-Taste dar: Wird sie gedrückt, so erfährt das Bertiebssystem davon durch eine andere Statusmeldung des POKEY und reagiert dann entsprechend durch die Break-Routine darauf.

Ist es jedoch eine der normalen Tasten, wird das Drücken vom POKEY dadurch bemerkt, daß er ständig eine Matrix abfragt (pollt), in deren Kreuzungspunkte die einzelnen Tastenschalter liegen. Ist nun ein solcher Taster geschlossen, bemerkt der POKEY dies und erkennt an seinen Registern, welcher Schnittpunkt, also welcher Schalter das ist.

Darüberhinaus besitzt er zwei Signalleitungen, die ihm mitteilen, wenn die SHIFT- oder CONTROL-Taste gedrückt ist. Alleine das Drücken dieser beiden Tasten löst noch keinen Interrupt aus, verändert jedoch bei einem richtigen Tastendruck dessen Kode.

Es sind maximal 64 Tasten im System erlaubt. Sie belegen somit die Kodes 0 bis 63 ($00 bis $3f). Ist die SHIFT-Taste gedrückt, verschiebt sich der generierte Kode um 64, also auf den Bereich 64 bis 127 ($40 bis $7f).

Im Falle der gedrückten CONTROL-Taste würde sich der Kode dann nochmal um 64 verschieben, also um insgesamt 128 auf den Bereich von 128 bis 191 ($80 bis $bf). Werden SHIFT- und CONTROL-Taste gleichzeitig gedrückt, hat die CONTROL-Taste Vorrang, so daß also insgesamt 192 Tastenkodes entstehen können.

Um nun diesen Matrix-Nummern einen sinnvollen ATASCII-Kode zuweisen zu können, wird eine Tabelle benutzt. Der Anfang der Tabelle liegt ab (KEYDEFPTR), also im Initialisierungszustand ab Adresse KEYDEF. Die 192 Byte große Tabelle besitzt die folgenden Einträge, bei denen nur diejenige Funktion eingetragen ist, die sich ohne SHIFT- oder CONTROL-Taste ergibt:

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$00:	L	J	;	/1/	/2/	K	+	*	O		P	U	/C/	I	-	=
$10:	V		C	/3/	/4/	B	X	Z	4		3	6	/E/	5	2	1
$20:	,	/S/	.	N		M	/	/A/	R		E	Y	/T/	T	W	Q
$30:	9		0	7	/B/	8	µ	°	F	H	D		/P/	G	S	A

/C/ ist die RETURN-Taste
/S/ ist die Leertaste
/A/ ist die ATARI-Logo-Taste
/E/ ist die ESCape-Taste
/T/ ist die TABulator-Taste
/B/ ist die BackSpace-Taste
/P/ ist die CAPS-Taste
/1/ bis /4/ sind die zwar nicht hardwaremäßig, aber dennoch softwaremäßig vorhandenen Funktionstasten des 1200XL F1 bis F4.

Ob die SHIFT-Taste gedrückt ist, besagt Bit 3 von SKSTAT. Ist das Bit Null, ist die SHIFT-Taste gedrückt.

Eine genauere Beschreibung des Registers SKSTAT erfolgt noch später.

```
***********************
*                     *
*                     *
*   PADDLESTEUERUNG   *
*   ---------------   *
*                     *
***********************
```

Paddles sind Eingabegeräte, die nicht nur digitale Signale liefern wie z.B. Joysticks, sondern die einen analogen Wert liefern, in diesem speziellen Fall handelt es sich um einen Widerstandswert.

Da der Atari bekanntlich nur digitale Signale richtig verarbeiten kann, stellt sich für ihn, genauer: Für den POKEY, das Problem, aus diesem Widerstandswert eine Zahl zu machen.

Diesen Vorgang nennt man schlichtweg Analog/Digital-Wandlung, was sich englischsprachig abgekürzt A/D-C schreibt, wobei das 'C' für Converting steht.

Dieser gesamte Komplex der Analog/Digital-Wandlung und der artverwandten Digital/Analog-Wandlung ist bis heute die große Schwachstelle an jedem Computersystem. Nicht, daß die Techniker nicht wüßten, nach welchem Verfahren man die Wandlung vornehmen soll. Da gibt es einige wenige, die sich als sinnvoll in den letzten Jahren herauskristallisiert haben.

Dagegen ist es wesentlich schwieriger, einen Kompromiß zu finden aus mittlerer Wandlungsgeschwindigkeit und Genauigkeit des Systems.

Es ist für einen elektronischen Schaltkreis relativ einfach, festzustellen, ob er größer oder kleiner/gleich einer Vergleichsspannung ist. Es gibt jedoch schon Justierungsaufgaben, wenn derselbe Baustein feststellen soll, ob sich die Eingangsspannung innerhalb eines bestimmten Bereichs, eines sogenannten Fensters, befindet.

Soll der Baustein nun herausfinden, in welchem der z.B. 4096 Fenster sich der Eingangsspannungswert befindet, dann ist das eine langwierige und schwierige Angelegenheit.

Der POKEY ist nun ein Baustein, der gleichzeitig acht solcher Kanäle überwacht und ausmißt. Damit er das einige Male pro Sekunde tun kann, betrachtet er nicht allzuviel mögliche Fenster, sondern nur 228 Stück. Die einzelnen Werte der POT-Eingänge (POT steht für Potentiometer, ein veränderbarer elektrischer Widerstand) können aus den Registern POT0 bis POT7 ausgelesen werden.

Es ist zu beachten, daß der 600XL/800XL nur insgesamt vier der acht möglichen Kanäle benutzen kann, da er nur über zwei Joystickbuchsen verfügt.

Beim Vergleich der Adressenliste für den POKEY fällt auf, daß die Adressen AUDI....n und POTn gleich sind. Das bereitet auch keinerlei Probleme, da die AUDI....n-Werte nur zu schreiben und die POTn-Werte nur zu lesen sind.

Die eigentliche Meßprozedur läßt sich über zwei mögliche Register starten. Das eine Register ist SKCNTL, in dessen Bit 2 festgelegt wird, ob es sich um eine normale oder eine sogenannte Schnellmessung handelt. Ist es eine Schnellmessung, so ist sie allerdings recht ungenau.

Das andere Register nennt sich POTGO und wird einfach mit irgendwelchen Daten beschrieben, um eine Messung zu initiieren.

Jeder Kanal besitzt ein Bit in dem Statusregister ALLPOT, das Auskunft darüber gibt, ob der Kanalwert des entsprechenden Kanals 'fertig' ist oder nicht. Ist das entsprechende Bit gesetzt, so gilt der Wert nicht, die Messung ist noch nicht fertig.

```
******************************
*                            *
*                            *
*   SERIELLE EIN-/AUSGABE    *
* -------------------------- *
*                            *
******************************
```

Unter serieller Ein-/Ausgabe versteht man das Senden und/oder Empfangen von Daten über hauptsächlich eine einzige Datenleitung. Die serielle Ein-/Ausgabe steht im Gegensatz zur parallelen Ein-/Ausgabe, bei der z.B. komplette Bytes auf nebeneinander verlaufenden elektrischen Leitungen oder anderen Informationsstrecken gesendet oder empfangen werden können. Solch eine parallele Übertragungsstrecke ist z.B. der vielleicht von Druckern her bekannte Centronics-Standard.

Steht zur Datenübertragung nur eine einzige Datenleitung zur Verfügung, so muß man sich überlegen, wie die Daten darüber zu transportieren sind.

Will man nur senden oder nur empfangen, ist schon ein Problem gelöst. Soll jedoch sowohl empfangen als auch gesendet werden können, stellt sich die Frage, ob man diesen Datentransport nur über eine Leitung laufen läßt (Halb-Duplex), oder ob dafür zwei Leitungen installiert werden (Voll-Duplex).

Wird das Halb-Duplex-Übertragungsverfahren gewählt, hat man das Problem, erkennen zu müssen, ob und welches der angeschlossenen Geräte gerade senden will.

Bei Verwendung des Voll-Duplex-Übertragungsverfahrens treten an Stelle der oben beschriebenen Übertragungsprobleme höhere Leitungskosten auf.

Da die Leitungskosten jedoch bei solch kurzen Verbindungen wie denen des Atari-Systems vernachlässigbar gering sind im Gegensatz zu dem Halb-Duplex-Umschaltaufwand, hat sich die Firma Atari für das für den Programmierer und Anwender wesentlich angenehmere Zwei-Leitungsverfahren entschieden.

Ist vom Entwickler diese Frage entschieden, stellt sich gleich die nächste: Synchrone oder asynchrone Übertragung?

Da es sich beim Atari jedoch im Prinzip um selten auftretende Übertragungsprozesse handelt, wird man hier gleich auf die asynchrone Übertragungsweise zurückgreifen.

Unter asynchroner Übertragung verstehen wir ein Verfahren, bei dem der Empfänger ständig darauf wartet, daß einmal ein Zeichen übertragen wird. Anfang und Ende eines solchen Zeichens sind durch ein, dem Zeichen vorangehendes, Startbit sowie durch ein, eineinhalb oder zwei Stopbits, die dem Zeichen folgen müssen, gekennzeichnet.

Richtig, das ist kein Druckfehler: Im Gegensatz zu der Auffassung, daß ein Bit die kleinste darstellbare Informationseinheit sei, gibt es nun bei der Datenübertragung auch halbe Bits. Wie dies zustandekommt, wird leicht klar, wenn man sich etwas näher ansieht, wie die Zeichen mit den Start- und Stopbits durch die Leitung kommen.

Bei diesem Übertragungsverfahren wird davon ausgegangen, daß immer nur genau ein Zeichen gesendet wird, das in Start- und Stopbits 'eingepackt' ist und außerdem vom Programmierer sowohl beim Empfänger als auch beim Sender die gleiche Übertragungsgeschwindigkeit eingestellt ist. Diese Geschwindigkeit ist das Her(t)z jeder seriellen Übertragung. Während

bei einer parallelen Strecke eine Leitung sagt "Hallo Empfänger, jetzt kommt ein Byte!" und der Empfänger nach Erkennen des Zeichens die Meldung bringt "Hallo Sender, das Zeichen wurde gelesen und verarbeitet", gibt sich hier diese Möglichkeit nicht, so daß man davon ausgehen muß, daß der Empfänger genau dann ein Bit zu lesen versucht, wenn es der Sender zur Verfügung stellt.

Genau diesen Effekt erreicht man durch den kurzzeitigen Gleichlauf von Sender und Empfänger, wenn man beiden mitteilt, daß jedes Bit eine ganz bestimmte Zeit anliegt. Da dieser Gleichlauf nur immer ein Zeichen lang existieren muß (denn dann kommt eine neue Synchronisation durch das nächste Startbit!), ist das technisch recht einfach lösbar.

Im Laufe der letzten 30 Jahre haben sich ganz bestimmte Übertragungsgeschwindigkeiten durchgesetzt. Von Grundgeschwindigkeiten ausgehend, hat man es durch immer bessere Technik immer wieder geschafft, die Übertragungsrate zu verdoppeln. So existieren heute die folgenden allgemein verwendeten Übertragungsraten

```
    45,45  Bit/Sekunde (nur in den USA)
    50     Bit/s      z.B. bei BTX
   100     "
   150     "
   300     "
   600     "
 1.200     "          z.B. bei BTX
 2.400     "
 4.800     "
 9.600     "
19.200     "
38.400     "
```

Höhere Übertragungsraten sind bei seriellen Schnittstellen sehr selten, da es dabei technische Probleme gibt.

Hat man es nun also geschafft, den Sender beziehungsweise Empfänger auf eine bestimmte Übertragungsrate einzustellen, so ist bekannt, wie lang ein Bit auf der Übrtragungsleitung anliegt:

Ist z.B. die Übertragungsrate auf 19.200 Bit/s eingestellt, ist jedes Bit 1/19.200 Sekunden, also ca. 52 millionstel Sekunden lang. Das ist nicht sehr viel, aber dem Computer genügt es.

Um nun ein Stopbit zu senden, wird einfach die Sendeleitung für die Dauer eines Bits in den Ruhezustand versetzt. Dieser Ruhezustand kann jedoch auch länger dauern, z.B. die 1,5-fache Dauer eines Bits; also 1,5 Stopbit!

Nach dieser etwas technischen Einleitung dürfte eine Zeichnung von zwei zu übertragenden Zeichen ganz hilfreich sein:

Es sollen die Bytes $53 und $8a übertragen werden mit einem Startbit und einem Stopbit:

$53 = %0101 0011
$8a = %1000 1010

f = Leitung im Ruhezustand (SPACE, frei)
S = Startbit, immer MARK
s = Stopbit, immer SPACE



ffff S 0 1 0 1 0 0 1 1 s ffff S 1 0 0 0 1 0 1 0 s ffff

Die Bereiche ffff können beliebig lang sein bei der asynchronen Datenübertragung (daher der Name: Asynchron, nicht zeitlich festgelegt).

Nachdem alle diese Dinge entschieden sind, ist man nun in der Lage, einzelne Zeichen (Bytes) zu übertragen, und zwar sowohl zu senden, als auch zu empfangen.

Um nun jedoch eine richtige Datenübertragung zustande zu bekommen, ist es weiterhin noch notwendig, sich ein sogenanntes Protokoll zu überlegen, nach dem die einzelnen Geräte an der seriellen Leitung angesprochen werden sollen und anhand dessen dann komplette Datenblöcke (durch zum Beispiel NEWLINE abgeschlossene Zeilen oder Blöcke fester Länge) gehandhabt werden können.

Bei Atari sind fast alle an der seriellen Leitung hängenden Geräte intelligent, das heißt, daß sie einen eigenen Kontrollbaustein besitzen, der die Kommunikation mit dem Hostsystem, also dem Atari 600XL/800XL übernimmt.

Das einzige nicht intelligente Gerät ist der Cassettenrecorder. Er wird vom Benutzer ein- und ausgeschaltet, wenn der Computer dies durch Huptöne empfiehlt.

Andere Geräte, wie zum Beispiel der Drucker oder die verschiedenen Diskettenstationen lauschen ständig die am seriellen Bus liegenden Informationen sowie die eine Leitung, genannt KOMMANDO:L, ab. Liegt KOMMANDO:L auf dem logischen Pegel Null, bedeutet dies, daß jetzt ein Gerät angesprochen werden soll. Dann sendet das Hostsystem die Geräteart-Kennung und die Gerätenummer sowie die Blocknummer bei Diskettenstationen zusammen mit dem Kommando über den Bus, um das richtige Gerät 'aufzuwecken'. Ist das adressierte Gerät vorhanden und bereit für die Berabeitung dieses Kommandos, so wird es sich zurückmelden mit einer Acknowledge-Meldung: Es sendet ein 'A' und eine kurze Statusinformation mit Checksumme. Ist das Gerät beziehungsweise das Kommando dagegen nicht in Ordnung, wird ein 'N' (=Negative Acknowledge) zurückgesendet.

Ist das Gerät bereit, beginnt der eigentliche Datentransfer, bei dem die zu schreibenden oder zu lesenden Zeichen im Block gesendet werden, wobei die KOMMANDO:L-Leitung wieder auf Eins gesetzt wird.

Sind alle Zeichen übertragen, oder glaubt zumindest der Sender, alle Zeichen übertragen zu haben, sendet er unter Umständen noch eine Prüfsumme. Ist das Peripheriegerät zum Beispiel eine Diskettenstation mit dem Befehl, einen bestimmten Block zu schreiben, so beginnt es mit der eigentlichen Arbeit (dem Schreiben des Blockes), nachdem es die zu schreibenden Daten vom Hostsystem erhalten hat. Stellt sich nun jedoch heraus, daß der zu schreibende Block defekt ist, sendet die Diskettenstation ein 'E' als Error-Meldung an das Hostsystem zurück. Geht die Operation im ganzen gut, sendet das Gerät ein 'C' für 'Completed' zurück.

Das Hostsystem wartet während der gesamten Zeit der Verarbeitung auf eine positive oder negative Rückmeldung des Gerätes. Erscheint sie nach einigen (im Normalfall 7) Sekunden immer noch nicht, wird die Operation vom Hostsystem dadurch abgebrochen, daß ein Timeout eintritt.

In diesem Fall wiederholt das Betriebssystem die Operation so oft, wie es der Wert in CRETRY vorgibt (im Normalfall noch 1 mal).

Es ist dann Aufgabe des Hauptprogramms, sicherzustellen, daß eine eventuelle Fehlermeldung über die Statusregister des CIO oder SIO richtig interpretiert wird.

Viel einfacher ist dies alles beim Cassettenrecorder.

Dabei werden die Daten einfach nur nach einer kurzen Wartezeit gesendet und davon ausgegangen, daß sie schon richtig ankommen werden. Beim Lesen wird allerdings eine eigentlich sehr sinnvolle, leider nur sehr ungenaue Kontrolle vollzogen: Die Lesegeschwindigkeit stellt sich automatisch auf die empfangene Datenrate ein. Dies wird dadurch realisiert, daß jeder Block auf der Cassette mit zwei ganz bestimmten Byte beginnen muß. Sie haben beide den Wert $aa, was binär %1010 1010 darstellt und sich an sich sehr gut für die Synchronisation eignet.

Wer das Cassettenformat einmal verbessern will, kann ausprobieren, ob die Übertragung sicherer wird, wenn nicht diese,

sondern zwei andere Byte verwendet werden: %1100 1100. Damit müßten sich mit entsprechender Softwareänderung wesentlich genauere Meßergebnisse herstellen lassen.

Außerdem gibt es bei der Cassette noch einem weiteren wesentlichen Unterschied im Übertragungsverfahren zu allen anderen bislang bekannten Geräten: Der Cassettenrecorder 'versteht' keine reinen Digitalsignale, sondern er benötigt zum korrekten Arbeiten zwei unterschiedliche Töne. Ein tieferer Ton aus Tonkanal 2 stellt die logische Null dar und ein höherer Ton aus Tonkanal 1 stellt demzufolge die logische 1 dar. Diese Töne haben gegenüber Logikpegeln den Vorteil, daß sie direkt auf dem Magnetband gespeichert werden können. Bei Pegeln (die sich theoretisch auch speichern lassen) gibt es bei Cassettenrecordern dieser Preisklasse und derart geringen Übertragungsraten erhebliche technische Schwierigkeiten. Nicht umsonst sind die echten Computerbandmaschinen immer noch recht große, teure Schränke, können dagegen jedoch auch in der Übertragungsrate leicht mit den heutigen Atari-Diskettensystemen konkurrieren.

Der POKEY hat nun bei alledem eine ganze Reihe von Aufgaben:

Erstens übernimmt er die Erzeugung der Sende beziehungsweise Lesetakte. Ist der POKEY der Erzeuger der Taktsignale, so gibt er sie über die Leitung CLOCK OUT aus.

Der Sendetakt kann durch die Taktrate in Kanal 2, Kanal 4 oder durch einen externen Takt über die Leitung CLOCK IN bestimmt werden.

Für den Lesetakt gilt das gleiche wie für den Sendetakt. Die Sendedaten wechseln jeweils bei der steigenden Flanke des Taktsignals, wohingegen die Lesedaten bei der fallenden Flanke gelesen werden. Dies soll sicherstellen, daß beim Lesen ein halber Takt gewartet wird, um dem Signal Zeit zum Einschwingen zu geben.

Als Zweites hält der POKEY die zu sendenden Daten im SEROUT und die gerade empfangenen Daten im SERIN Register für die weitere Verarbeitung fest.

Wird in das SEROUT-Register ein Wert geschrieben, so wird er sofort in das parallel/seriell-Wandlungsregister geschrieben, wenn dieses frei ist. Danach gibt der POKEY einen Interrupt, in dem er ankündigt, daß das SEROUT-Register leer sei, und der POKEY damit bereit sei, ein neues Zeichen anzunehmen, obwohl der Baustein selbst noch mit dem Senden des alten Datums beschäftigt ist.

Wird kein neuer Wert in das SEROUT-Register geschrieben, erfolgt nach dem Entleeren des parallel/seriell-Wandlungsregisters ein weiterer Interrupt, der angibt, daß die Übertragung des letzten Zeichens beendet sei.

Beim Lesen gibt der POKEY einen Interrupt, wenn er ein komplettes Zeichen gelesen hat. Das Zeichen übergibt er dann im SERIN-Register und setzt jedoch noch einige Statusbits im SKSTAT-Register. Es kann zum Beispiel passieren, daß der Computer nicht rechtzeitig die in SERIN stehenden Daten liest, und der POKEY ein neues Byte empfängt, nach SERIN schreibt und somit die alte Information überschreibt. In diesem Fall ist das Bit 6 'serieller Überlauf' gesetzt.

Stimmt dagegen etwas mit den Start/Stopbits des Zeichens nicht, so wird das 'Framing-Fehler'-Bit Nummer 7 gesetzt. Dieser Framing- (Rahmen-) Fehler kann auftreten, wenn zum Beispiel ein BREAK-Zeichen über die serielle Schnittstelle gesendet wird. Dies ist kein eigentliches Zeichen, sondern ein Synchronisations- und Resetmechanismus, bei dem die Sendeleitung für ungefähr eine viertel Sekunde auf LOW (also aktiv, MARK) gezogen wird. Dabei wird der POKEY natürlich ein Zeichen $00 erkennen, danach jedoch die beiden Stopbits vermissen.

Will man genau wissen, welchen Pegel die Sende/Empfangsleitung besitzt, so kann man Bit 4 von SKSTAT testen.

Dieses Bit ist eine Hardwarekopie des Eingangssignals und bietet neben dieser (noch nicht programmierten) Möglichkeit der BREAK-Erkennung diejenige, die Übertragungsgeschwindigkeit festzustellen, ohne direkt die seriell/parallel-Wandlung des POKEY initialisieren zu müssen.

Ist einmal ein Fehler aufgetreten, so wird das entsprechende Bit dadurch zurückgesetzt, daß ein beliebiger Wert in SKRESET geschrieben wird.

Die einzelnen Bits des SKSTAT haben die folgende Bedeutung:

Bit		Bedeutung
7	MSB	Framing-Fehler bei der seriellen Übertragung
6		Overrun-Fehler bei der seriellen Übertragung. Es wurde wenigstens ein Zeichen verloren.
5		Overrun-Fehler bei der Tastatureingabe. Es wurde wenigstens ein Tastendruck verloren.
4		Dies ist die direkte Hardwarekopie des seriellen Eingangs. Dieses Bit wird nur für Spezialanwendungen wie Synchronisation abgefragt.
3		Die SHIFT-Taste auf der Tastatur ist gedrückt.
2		Die letzte Taste ist immer noch gedrückt
1		Das Schieberegister der parallel/seriell-Wandlung arbeitet immer noch, das heißt, die Datenübertragung ist noch nicht abgeschlossen. Dieses Bit sagt zwar nichts darüber aus, ob nicht ein weiteres Datenbyte in das SEROUT-Register geschrieben werden darf, aber wenn man keine genauen Informationen über den Status der seriellen Ausgabe hat, so sollte man unbedingt das Einswerden dieses Bits abwarten.
0	LSB	Dieses Bit ist nicht benutzt und ist immer auf 1.

MSB = Most Significant Bit = höchstwertiges Bit
LSB = Least Significant Bit = geringstwertiges Bit

Die Bits in SKSTAT sind im Ruhezustand immer auf Eins und besitzen die angegebene Aussage, wenn sie auf Null gehen.

Das Register SKCNTL beschreibt neben den Übertragungsgeschwindigkeiten noch die folgenden Funktionen:

Bit		Bedeutung
7	MSB	Abbrechen der seriellen Ausgabe und Senden von SPACE bei Eins. Dient zum Initialisieren der Sendeleitungen und Senden von mehr als einem Stopbit.
6		Die drei Bit dienen der Festlegung
5		der Übertragungsgeschwindigkeiten
4		des Senders und Empfängers.
3		Wird dieses Bit auf Eins gesetzt, erfolgt die Ausgabe der Bits im Zweitonverfahren.
2		Durch das Setzten dieses Bits wird der Schnellgang der Paddlesteuerung eingeschaltet. Die Messung erfolgt dann nicht wie üblich innerhalb von maximal 20ms, sondern in der Zeit von nur zwei Bildzeilen (ca. 128us).
1		Dieses Bit wird für die Tastaturverarbeitung intern benutzt.
0	LSB	Sind Bit 1 und Bit 0 auf Null, so erfolgt ein Software-Reset des POKEY.

Das SKCNTL hat ein Schattenregister SKCNTL§.

Die Kombinationsmöglichkeiten der drei Bits 4, 5 und 6 des SKCNTL lassen sechs technisch mögliche Zustände zu:

Bit 7	6	5	Bedeutung
0	0	0	Alle Takte werden von außen bestimmt. Die internen Takte gehen auf Null.
0	0	1	Die Senderate wird durch einen externen Takt bestimmt, die Eingangsrate wird von Kanal 4 (oder dem 16bit-Zähler, bestehend aus Kanal 3 und Kanal 4) geliefert. Es erfolgt asynchrones Lesen der Daten.
0	1	0	Sowohl die Sende- als auch die Empfangsfrequenzen werden von Kanal 4 festgelegt.
0	1	1	Nicht nutzbar, da der Ausgangstakt sich beim Lesen eines Zeichens verschieben würde (Synchronisation asynchroner Daten).
1	0	0	Die Senderate liegt bei diesem Verfahren durch die Programmierung des Kanals 4 fest, die Eingangsrate wird durch einen externen Takt bestimmt.
1	0	1	Auch diese Taktform ist nicht nutzbar.
1	1	0	Hier bestimmt Kanal 2 die Senderate und Kanal 4 die Empfangssrate. Der Takt von Kanal 4 liegt an der externen Taktausgangsleitung an.
1	1	1	Die Ausgangsrate wird wie bei 110 durch Kanal 2 bestimmt, die diesmal asynchrone Leserate durch Kanal 4. Der Taktein-/Ausgang ist nicht benutzt und liegt offen.

Die Verfahren 110 und 111 sind zwar die komplexesten, aber sie beinhalten eine Einschränkung: Da zur Taktbestimmung Kanal 2 benutzt werden muß, kann hier keine Zweitonübertragung stattfinden!

Als letztes Register des POKEY ist das Interruptregister zu besprechen.

Auch dieses Register ist in zwei Teile aufgespalten: Ein Leseregister IRQSTAT und das Schreibregister IRQEN.

Gibt der POKEY einen der unten beschriebenen Interrupts an die CPU, so prüft deren Interrupthandler über IRQSTAT, welche der Interruptquellen es genau war und verzweigt zu den entsprechenden Routinen.

Sind alle Bits in IRQSTAT auf Eins, so ist der POKEY nicht die Unterbrechungsquelle.

Bit		Bedeutung (IRQSTAT)
7	MSB	Unterbrechung aufgrund BREAK-Tastendruck
6		Irgend eine andere Taste ist gedrückt
5		Der serielle Eingang hat Daten gelesen und sie in SERIN bereitgestellt
4		Das letzte in SEROUT eingetragene Zeichen wurde in das POKEY-interne parallel/seriell-Wandlungsregister übertragen und es kann das nächste Zeichen nach SEROUT transportiert werden.
3		Das parallel/seriell-Wandlungsregister des POKEY ist leer und in das SEROUT-Register wurde auch kein neuer Wert eingetragen. Die Übertragung ist somit für den POKEY beendet.

Bit		Bedeutung (IRQSTAT) (Fortsetzung)
2		Unterbrechung durch Nullwerden von Timer 4
1		Unterbrechung durch Nullwerden von Timer 2
0	LSB	Unterbrechung durch Nullwerden von Timer 1

Sämtliche Interrupts können einzeln erlaubt oder disabled werden. Eine Unterbrechung durch ein bestimmtes Ereignis ist dann erlaubt, wenn das zu dem Interrupt gehörige Bit gesetzt (auf 1) ist. Ein in IRQSTAT auf Null stehendes Bit wird wieder auf 1 gesetzt, wenn das Pendant in IRQEN ebenfalls zumindest kurzzeitig auf Null gesetzt wird.

Die Zuordnung der Bits zu den Interruptquellen ist bei IRQSTAT und IRQEN gleich:

Bit		Bedeutung (IRQEN)
7	MSB	Unterbrechung aufgrund BREAK-Tastendruck erlaubt
6		Beliebiger Tasteninterrupt erlaubt
5		Serielle Eingangsunterbrechung erlaubt
4		SEROUT-Interrupt enabled
3		Übertragungsendunterbrechung gestattet
2		Unterbrechung durch Nullwerden von Timer 4 erlaubt
1		Unterbrechung durch Nullwerden von Timer 2 erlaubt
0	LSB	Unterbrechung durch Nullwerden von Timer 1 erlaubt

Zu beachten ist auch hier das Schattenregister für IRQEN, IRQEN§. Wird ein Interrupt neu erlaubt oder verboten, so ist dies sowohl im Schatten- als auch im Hardwareregister zu tun. Ist die Abschaltung nicht so eilig, kann sie also im nächsten Vertical Blank Interrupt geschehen; so muß nur das Schattenregister korrigiert werden.

Die Korrektur erfolgt durch 'EinODERn' beziehungsweise 'Heraus-UNDen' des oder der jeweiligen Bits.

D I E   P I A

Die PIA wird im Atari hauptsächlich als Peripheriebaustein für die Joystickeingänge verwendet. Zudem steuern einige Ausgänge der PIA die MMU des Atari 600/800XL. Bei den alten Atari-Geräten werden diese Ausgänge für den 3. und 4. Joystickanschluß verwendet. Ansonsten übernehmen weitere Anschlüsse der PIA die Steuerung des seriellen Ports.

Die PIA ist im Gegensatz zu Bausteinen wie dem ANTIC oder dem GTIA kein Spezialbaustein, sondern ein Standardperipheriebaustein (6520 oder 6820) der 65XX- bzw. 68XX-Reihe.

Sie verfügt über zwei unabhängige 8-bit breite Ports (mit den Leitungen PA0-7 und PB0-7), bei denen sich jedes einzelne Bit über sogenannte Datenrichtungsregister als Ein- oder Ausgang programmieren läßt. Für jeden der Ports sind zwei Kontrollein- bzw. -ausgänge vorhanden (CA1, CA2, CB1 und CB2). Die beiden "Hälften" der PIA beeinflußen sich grundsätzlich nicht und sind, abgesehen von kleinen Unterschieden in den elektrischen Eigenschaften der Ports (die jedoch nur für Schaltungsentwickler wichtig sind) und Unterschieden in der Behandlung von CA2/CB2, identisch. Über die Kontrollkanäle ist es möglich, Interrupts der CPU auszulösen. Die Kontrollkanäle sind ursprünglich für das sogenannte "Handshaking", das heißt für die Synchronisation der Datenübertragung der PIA-Ports gedacht. Da sie aber universell ausgelegt sind, konnten sie von Atari auch für andere Zwecke verwendet werden. Die Ports der PIA werden nur für die Joysticks und die MMU, die kein Handshaking benötigen, benutzt.

Es folgt eine Tabelle der Belegung der PIA-Portleitungen:
(Die Angaben von PIN-Nummern beziehen sich jeweils auf die Steckverbinder und nicht auf die PIA-PINs)

PORT A :

PA0	Joystick-Port 1,	vorwärts	(PIN 1)
PA1	Joystick-Port 1,	rückwärts	(PIN 2)
PA2	Joystick-Port 1,	links	(PIN 3)
PA3	Joystick-Port 1,	rechts	(PIN 4)
PA4	Joystick-Port 2,	vorwärts	(PIN 1)
PA5	Joystick-Port 2,	rückwärts	(PIN 2)
PA6	Joystick-Port 2,	links	(PIN 3)
PA7	Joystick-Port 2,	rechts	(PIN 4)

CA1	SIO-Anschluß "Proceed" (PIN 9)
CA2	SIO-Anschluß "Motor-Kontrolle" (PIN 8)

PORT B :

PB0	MMU (ROM/RAM, RAM im Betriebssystem-Bereich)
PB1	MMU (BasEn, Basic einschalten)
PB2-6	nicht verwendet
PB7	MMU (TEST, Selbsttestprogramm einblenden)

CB1	SIO-Anschluß "Interrupt" (PIN 13)
CB2	SIO-Anschluß "Kommando" (PIN 7)

Bei den alten Atari-Geräten ist Port B mit Joystick 3 und 4 wie Port A mit Joystick 1 und 2 belegt.

Die PIA belegt im System mindestens 4 Bytes. Daß sie im Atari erheblich mehr belegt, liegt an der unvollständigen Adressdekodierung. Jede PIA-Hälfte hat also zwei Adressen. Diese nennen sich PORTA, PORTACNTL, PORTB und PORTBCNTL. PORTACNTL und PORTBCNTL steuern die Funktionen der PIA. Außerdem wird durch sie entschieden, ob man in die Adressen PORTA und PORTB zu übertragende Daten schreibt oder über diese Adressen die Datenrichtungsregister programmiert.

Jedes Bit in einem der Datenrichtungsregister repräsentiert ein Bit des jeweiligen Ports. Ist das entsprechende Bit des Datenrichtungsregisters "0", so fungiert das Bit des Ports

als Eingang. Dementsprechend dient das selbe Bit als Ausgang, wenn das dazugehörige Bit des Datenrichtungsregisters high ist.

Es folgt die Beschreibung der Funktion der einzelnen Bits von Register PORTACNTL:

Bit 0 und 1	CA1 Steuerung
Bit 2	wenn "0" dann wird über Adresse PORTA das Datenrichtungsregister A angesprochen
Bit 3 bis 5	CA2 Steuerung
Bit 6	IRQA1 beim Lesen des Registers PORTACNTL ist dieses Bit "1", wenn zuvor die Interruptbedingung an CA2 vorlag. Beim Lesen des Portregisters wird dieses Bit auf "0" zurückgesetzt.
Bit 7	IRQA2 wie Bit 6, aber anstelle von CA2 wird dieses Bit von CA1 gesteuert.

Die Steuerung von CA1 erfolgt, wie aus der Tabelle ersichtlich, durch Bit 0 und Bit 1 von PORTACNTL:

Bit 1	Bit 0	Beschreibung
0	0	Bei der fallenden Flanke von CA1 wird Bit 7 des PORTACNTL "1". Der Interruptausgang A der PIA bleibt auf logisch "1".
0	1	Bei der fallenden Flanke von CA1 wird Bit 7 des PORTACNTL "1". Der Interruptausgang A der PIA geht auf logisch "0" und fordert somit einen IRQ von der CPU.
1	0	Bei der steigenden Flanke von CA1 wird Bit 7 des PORTACNTL "1". Der Interruptausgang A der PIA bleibt auf logisch "1".
1	1	Bei der steigenden Flanke von CA1 wird Bit 7 des PORTACNTL "1". der Interruptausgang A der PIA geht auf logisch "0" und fordert somit

einen IRQ von der CPU.

CA2 Steuerung:

Ist Bit 5 des PORTACNTL low, so dient CA2 ebenfalls als Interrupteingang. Dabei gilt die gleiche Tabelle wie für die Steuerung von CA1. Statt Bit 0 und Bit 1 kontrollieren Bit 3 und 4 des PORTACNTL-Registers die Funktion. In die Tabelle ist statt Bit 7 des PORTACNTL, Bit 6 einzusetzen.
Ist Bit 5 des PORTACNTL high, so dient CA2 als Ausgang. Es gilt dabei folgende Tabelle:

Bit 4	Bit 3	Beschreibung
0	0	CA2 wird bei der ersten negativen Flanke des Systemtaktes nach einer Leseoperation von PORTA low und wieder high, wenn Bit 7 des PORTACNTL durch CA1 gesetzt wird.
0	1	CA2 wird bei der ersten negativen Flanke des Systemtaktes nach einer Leseoperation von PORTA low und bei der ersten negativen Flanke des Systemtaktes, bei der die PIA nicht angesprochen wird, wieder high.
1	0	CA2 bleibt low solange dieses Bitmuster anliegt.
1	1	CA2 bleibt high solange dieses Bitmuster anliegt.

Die Steuerung des Port B ist im Aufbau gleich der des Port A. Alle Tabellen mit Ausnahme der letzten können übernommen werden. Jedoch muß PORTBCNTL anstelle von PORTACNTL betrachtet werden. Statt IRQA muß es IRQB heißen, statt CA1 und CA2 entsprechend CB1 und CB2 und statt des Zugriffs auf das Datenrichtungsregister A steuert Bit 2 den Zugriff auf das Datenrichtungsregister B. Interrupts, die von CB1 bzw. CB2 ausgehen, werden der CPU über Interruptausgang B übermittelt.

Wenn Bit 5 des PORTBCNTL high ist, dient CB2 als Ausgang. Dabei gilt folgende Tabelle:

Bit 4	Bit 3	Beschreibung
0	0	CB2 wird bei der ersten positiven Flanke des Systemtaktes nach einer Schreiboperation auf PORTB low. Wird Bit 7 von außen durch CB1 gesetzt, wird CB2 wieder high.
0	1	CB2 wird bei der ersten positiven Flanke des Systemtaktes nach einer Schreiboperation auf PORTB low, und bei der ersten positiven Flanke des Systemtaktes, bei der die PIA nicht angesprochen wird, wieder high.
1	0	CB2 bleibt low, solange dieses Bitmuster anliegt.
1	1	CB2 bleibt high, solange dieses Bitmuster anliegt.

Auf den ersten Blick erscheinen die Steuerungsmöglichkeiten der PIA sicherlich unübersichtlich. Bei Überlegungen sollte man jedoch beachten, daß Port A ursprünglich als Eingang und Port B als Ausgang gedacht war. Bei genauerem Betrachten der Steuermöglichkeiten wird der Sinn der diversen Möglichkeiten der Kontrollkanäle CA1, CA2, CB1 und CB2 deutlich. Sie ermöglichen eine Verbindung einer PIA mit anderen Peripheriebausteinen, bei der jeweils die CPU ein Signal erhält (Interrupt), wenn die ausgegebenen Daten vom Empfänger verarbeitet sind oder neue Daten zum Einlesen durch den Prozessor zur Verfügung stehen. Die Interrupts lassen sich zudem abschalten. Die CPU kann trotzdem abfragen, ob die Unterbrechungsbedingung erfüllt ist. Sie kann auch beim Auftreten eines Interrupts die Herkunft des selben durch Abfragen von Bit 6 und 7 von PORTACNTL bzw. PORTBCNTL lokalisieren. Außerdem lassen sich die Kontrollkanäle direkt vom Peripheriegerät beeinflußen, dies geschieht natürlich erheblich schneller als eine Steuerung der Kontrollkanäle von der CPU

aus. Damit läßt sich die Datenübertragungsrate erheblich erhöhen. Mit diesen Möglichkeiten der PIA läßt sich, wie schon erwähnt, ein automatischer "Handshake" zwischen Peripheriegerät und Hauptgerät aufbauen.

Im Atari-System wird die PIA nicht zur Datenübertragung benutzt. Bei der Programmierung der PIA ist darauf zu achten, andere PIA-Anwendungen nicht zu beeinflussen. Bei Fehlprogrammierungen der PIA ist es zum Beispiel möglich, daß der serielle Port gestört wird, da die Steuerung des seriellen Ports über die PIA erfolgt. Eine weitere Gefahr stellt die MMU dar. Bei falscher Programmierung der PIA wird der Rechner umweigerlich durch die falsche MMU-Steuerung abstürzen. Auch hier ist besondere Vorsicht geboten.

Die PIA-Register befinden sich an folgenden Adressen:

PORTA	54016	$d300
PORTB	54017	$d301
PORTACNTL	54018	$d302
PORTBCNTL	54019	$d303

Wer die PIA bereits aus anderen Mikrocomputersystemen kennt, wird sich über die Reihenfolge der PIA-Register wundern. Die normale "Reihenfolge" der Register ist bekanntlich

PORTA PORTACNTL PORTB PORTBCNTL

Bei den Atari-Geräten existiert jedoch die oben beschriebene Reihenfolge, da die beiden Adressleitungen für die Register der PIA vertauscht sind.

Um die Stellung der Joysticks festzustellen, muß der Inhalt der Register PORTA und PORTB gelesen werden. Die Auswertung der Register ist jedoch gerade für Anfänger nicht ganz

einfach. Daher erstellt das Betriebssystem während der Vertikalsynchronisation Speicherstellen, die die Stellung der Joysticks repräsentieren (diese Register sind letztlich Schattenregister der Register PORTA und PORTB) :

Joystick1	632	$278	
Joystick2	633	$279	
(Joystick3)	634	$27A	nur bei Atari 400/800 !
(Joystick4)	635	$27B	nur bei Atari 400/800 !

Diese Speicherstellen enthalten die Stellung der Joysticks nach folgendem Muster:

```
          14
     10         6
11        15          7
     9          5
          13
```

D A S B E T R I E B S S Y S T E M D E S A T A R I

Das Betriebssystem eines Computers kann, unabhängig von Größe, Art, Alter und Aufbau des Rechners, als die Schnittstelle zwischen der Hardware, also der Elektronik, und der Anwendersoftware, also zum Beispiel Programmiersprachen, Text- und Datenverarbeitungsprogrammen und anderer Software bezeichnet werden. Es ist unter anderem dafür verantwortlich, dem Anwender (beziehungsweise Programmierer) definierte Softwarehilfsmittel zur Verfügung zu stellen, mit denen er zum Beispiel ein einzelnes Zeichen auf den Bildschirm oder ein anderes Peripheriegerät bringen oder von ihm holen kann. (Peripherie = griech. Umfeld, also alles 'Drumherum'. In der Computerbranche Ausdruck für Ein-/Ausgabegeräte wie zum Beispiel Terminals oder Drucker).

Außerdem ist das Betriebssystem diejenige Software, die das System nach dem ersten Einschalten in einen Grundzustand bringt. Diesen Vorgang nennt man allgemein Power-Up, was aussagen soll, daß hier die Versorgungsspannung das erste Mal seit dem letzten Ausschalten angelegt wird. Der dabei von dem Computersystem durchlaufene Zyklus wird als Power-Up-Reset bezeichnet.

Während dieses Power-Up-Resets werden sämtliche vom Betriebssystem benötigten Speicherstellen sowie die Ein-/Ausgabebausteine initialisiert.

Gerade auf diesen letzten Punkt, die Ein-/Ausgabebausteine, wird bei heutigen Betriebssystemen gesteigerte Aufmerksamkeit gerichtet. So war es noch vor wenigen Jahren üblich, jeden Datentransfer zum Beispiel vom Speicher auf einen Magnetträger wie eine Floppy-Disk, direkt von der CPU aus zu gestalten und somit Rechenzeit damit zu 'verbraten'. Heute gibt es genügend sogenannte Controller, die nach kurzem Anstoßen durch die CPU den Datentransfer und die Gerätekontrolle alleine übernehmen, so daß das eigentliche System nach Möglichkeit nicht weiter behindert wird.

So kann ein Controller damit beschäftigt sein, auf Anforderung eines zweiten, anderen Controllers, ein (das nächste) Datenelement aus dem Speicher auszulesen und eben dieses Datum an den zweiten Controller zu übergeben.

Dieser zweite Controller könnte dann die Aufgabe haben, dieses Zeichen nach ganz bestimmter Vorgehensweise auf die Floppy-Disk zu schreiben und diesen Vorgang auf Korrektheit hin zu überprüfen.

Solche Vorgehensweisen werden im allgemeinen DMA-Prozesse genannt, was Direkt-Memory-Access, direkter Speicherzugriff ohne die Benutzung der CPU bedeutet.

Der DMA hat den schon oben erwähnten Vorteil, daß bei gut ausgeführtem DMA-Konzept die eigentliche Rechenzeit nicht weiter beeinflußt wird, hat jedoch die Nachteile, daß die Controller aufgrund der beinhalteten Intelligenz nicht eben billig sind. Außerdem ist ein System, bei dem alle Aufgaben von nur einem Baustein bearbeitet werden, wesentlich leichter zu überwachen als eines, bei dem sich diverse, parallel arbeitende, 'Intelligenzbestien' gegenseitig beeinflussen und Fehler 'zuschieben'. Es gilt also leider auch in diesem Computerbereich der Satz, daß "viele Köche den Brei verderben".

Es gibt jedoch noch eine dritte Möglichkeit, ein System aufzubauen. Dabei werden, ähnlich dem DMA-Konzept, Intelligenzen 'verteilt'. Es existieren also auch intelligente Geräte, wie zum Beispiel eine Diskettenstation. Der Unterschied zum DMA-Betrieb besteht jedoch darin, daß die Diskette nicht direkt im Speicher des Hauptsystems 'herumfummelt', sondern nur über eine sogenannte Interruptleitung dem Haupt- oder auch Hostsystem mitteilt, daß es neue Daten braucht oder abgeben will. Es ist dann alleinige Aufgabe des Hostsystems, herauszufinden, von wem und auf welche Weise der Interrupt behandelt werden soll.

Im Normalfall wird solch ein Interrupt zuerst die CPU von der Meldung unterrichten. Dies geschieht dadurch, daß die CPU das eigentlich gerade bearbeitete Programm unterbricht, um einen sogenannten Interrupt-Handler aufzurufen. In diesem Interrupt-Handler, der meist Teil des Betriebssystems ist, stehen dann die, beim Auftreten eben dieser Bedingung auszuführenden, Kommandos. Damit wird dann entweder ein Baustein programmiert, der die beim unterbrechenden Gerät anliegenden

Daten einliest und verarbeitet (beziehungsweise an ein anforderndes Gerät übergibt) oder die CPU übernimmt selbst die Datenverarbeitung, um danach zu dem unterbrochenen Benutzerprogramm zurückzukehren.

Bei annähernd allen Geräten die es heutzutage verdienen, Computer genannt zu werden, treffen Sie Mischformen der drei oben angegebenen Konzepte an. Das gilt sowohl für den Hobbycomputermarkt als auch für die wesentlich teureren Bürogeräte.

Jeder Entwickler hat seine ganz perönlichen Einstellungen zu bestimmten Systemkonzepten, Vorlieben und Abneigungen zu einzelnen Bausteinen (oder Bausteinherstellern!), so daß man nicht von 'dem Computerkonzept' sprechen kann und dies wohl auch nie können wird.

Wie nach diesen einleitenden Erklärungen nicht anders zu erwarten, ist also auch das Atari 600XL/800XL-System eine Mischform aller drei Konzepte.

Als Bespiel für das reine CPU-Konzept kann die Tastaturabfrage dienen. Der Tastenkode wird zwar über einen Baustein (POKEY) eingelesen, dieser dient aber lediglich in diesem Falle als Pufferbaustein. Die eigentliche Dekodierarbeit, welcher Tastenkode zu welchem ATASCII-Zeichen gehört, wird von der CPU selbst übernommen.

Das DMA-Konzept ist, wie den entsprechenden Kapiteln zu entnehmen, in technisch ausgesprochen gut ausgereifter Weise beim ANTIC und GTIA enthalten. Der ANTIC versorgt unter anderem als intelligenter DMA-Controller den GTIA mit Daten.

Aber auch die dritte Systemart ist vorhanden: Die Atari 400/800- und 600XL/800XL Modelle verfügen über ein hohes Maß an Interrupts, von denen ein nicht unerheblicher Teil für die Datenübermittlung von oder zu peripheren Geräten über die serielle Schnittstelle und den POKEY-Baustein verwendet wird.

Dabei kann man grundsätzlich sagen, daß das System für einen Computer dieser Preisklasse über ein erstaunliches Niveau verfügt. Nicht unbedingt wegen der grundlegenden Konzeption; man findet sie in ähnlicher Form sicher auch bei Konkurrenzprodukten. Es sind jedoch zwei andere, für den reinen Benutzer nicht unmittelbar zu entdeckende Vorteile, die nur dann auffallen, wenn sie nicht vorhanden sind:

Zum einen ist das Gesamtsystem abgerundet und die einzelnen Komponenten jeweils sauber aufeinander abgestimmt. Dies und anderes stellt sicher, daß das System nach Möglichkeit jeden erdenklichen Fehler beim gleichzeiten Ausführen unterschiedlicher Aktivitäten 'abfängt' und korrigiert. Es gibt genug Computer, die pfiffige Grundelemente beinhalten. Kommt jedoch solch ein Element einmal ein wenig 'aus dem Tritt', gibt es keine Kontrollstuktur, die den Fehler auffängt.

Ein konkretes Beispiel hierzu wäre in den seriellen Schnittstellenroutinen zu finden. Vielleicht ist Ihnen schon einmal aufgefallen, daß der Atari beim Lesen oder Schreiben von/auf Diskette manchmal 'einschläft', also einige Sekunden lang nichts tut, um danach, wie von Geisterhand, völlig korrekt weiterzuarbeiten. Dieser Effekt tritt besonders häufig dann auf, wenn man Diskettenstationen fremder Hersteller verwendet.

Die Ursache des Fehlers ist in der Art der Kommunikation zwischen Computer und Diskettenstation zu suchen. Der Atari wartet nach dem Senden eines Kommandos auf eine Antwort des angesprochenen Gerätes. Kommt die Antwort jedoch etwas zu schnell für den Atari, weil dieser zum Beispiel gerade mit der Bearbeitung von Interrupts beschäftigt war, wertet er diese Antwort als fehlerhaft und wartet auf eine korrekte Meldung. Diese kann natürlich nicht kommen, weil die Diskettenstation der berechtigten Ansicht ist, schon geantwortet zu haben und vielleicht sogar nun selbst auf Daten wartet.

Dieser Konflikt, bei dem also zwei aufeinander wartende Geräte sich gegenseitig blockieren, wird Deadlock genannt und ist eine bei Betriebssystemtechnikern gefürchtete Erscheinung immer komplexer werdender Programme und führt bei

manchen Hobbycomputern zum 'Absturz', da der Konflikt nicht erkannt wird. Nicht jedoch beim 'Nobel-Betriebssystem' von Atari: Hier sorgt ein sogenannter Timeout dafür, daß die Übertragung weitergeht. Die CPU bekommt hierdurch mitgeteilt, daß die Übertragung offensichtlich 'hängt' und versucht sie ein zweites Mal.

Der zweite Vorteil des Atari-Betriebssystems gegenüber denen anderer Hersteller ist schlichtweg der, daß darin Ordnung herrscht.

Das bedeutet, daß das ganze System in kleine, überschaubare Routinen aufgeteilt ist, von denen normalerweise ganz klar definiert werden kann, was sie als Ausgabe bei welcher Eingabe liefern. Es ist, in den hier im Buch beschriebenen 12 KByte, mit einer Ausnahme davon abgesehen worden, 'Spaghetti-Kode' zu schreiben, also endlose Routinen, in denen alles und nichts getan wird. Die eine Ausnahme läßt sich ebenfalls begründen; es handelt sich dabei um eine Interruptroutine und dort wollte Atari einfach Bearbeitungszeit sparen.

Mit wenigen Ausnahmen kann man sagen, daß dieses Betriebssystem nicht von irgendwelchen Hackern geschrieben wurde, sondern von Ingenieuren, die Ihre Arbeit hervorragend verstehen.

So sind zum Beispiel die Ein-Ausgaberoutinen nicht für jeden Baustein anders, sondern es gibt sogenannte Input-Output-Control-Blocks (IOCBs), die die Art des Gerätes, das Kommando und alle damit verbundenen Werte definieren. Die Ausgabe findet nun dadurch statt, daß an die Ausgaberoutine nur noch die Nummer des IOCB übergeben zu werden braucht, und schon 'weiß' der Computer, wie er welches Gerät zu behandeln hat und welche Ein- bzw. Ausgabeoperation ausgeführt werden soll.

Diese Ordnung ist auch Grundlage dafür, daß Atari nicht schon zu Laufzeiten der alten 400/800-Serie diverse Betriebssysteme hervorbrachte, die dann nicht zueinander gepaßt hätten.

Atari hat zwar eine neue Version herausgebracht, in der einige Fehler verbessert wurden, es muß jedoch klar gesagt werden, daß Software, die für den 400/800 geschrieben wurde, voll auf den neuen Systemen lauffähig ist, wenn nur die, den autorisierten Händlern und Softwareproduzenten bekannten und von denen vielfach publizierten, Originaleinsprungadressen verwendet wurden.

Es dürfte auf der Hand liegen, daß ein Pogramm, in dem 'raffiniert getrickst' worden ist, natürlich an eine bestimmte Betriebssystemversion gebunden ist. Ein vernünftig, also ingenieurmäßig, arbeitender Programmierer hat dagegen beim Systemwechsel bei Atari keinerlei Probleme.

Dazu trägt auch noch bei, daß bei Atari dieses Betriebssystem wirklich nur eben das Betriebssystem ist, und nicht zum Beispiel Verquickungen mit einer eingebauten Programmiersprache, wie etwa BASIC, programmiert sind. Da kann es in anderen Systemen zum Beispiel vorkommen, daß in einem BASIC-Interpreter Routinen für Peripherieschnittstellen definiert sind. Wird also das BASIC abgeschaltet, darf man sich diese Schnittstelle neu programmieren.

Alle diese Mankos sind hier beim Atari 600XL/800XL weitestgehend vermieden worden. Zwar ist im Rechner ein Mathematik-ROM fest eingebaut, es wird jedoch nicht (wirklich!) vom Betriebssystem benutzt.

Die Einschränkung 'wirklich' muß gemacht werden, da in der normalen Systemkonfiguration bei bestimmten Ein-/Ausgabeoperationen plötzlich bestimmte Stellen in diesem ROM angesprochen, beziehungsweise direkt angesprungen werden. Das heißt jedoch nicht, daß dort gerechnet werden soll, sondern daß dieses ROM einmal von irgendwelcher Periherie abgeschaltet und der dann freigewordene Platz mit neuem Programmspeicher belegt werden soll. In diesem Adreßbereich könnten dann zusätzliche Treiberprogramme liegen.

Welcher Art diese Peripherie sein soll, läßt sich nur erahnen, da uns außer der 64 KByte-RAM-Karte beim 600er keine

Geräte bekannt sind, die auf dem hinten anliegenden Systembus angeschlossen würden. Nur in diesem Fall nämlich könnte das Peripheriegerät den im Mathematik-ROM liegenden Adreßbereich füllen.

Ganz allgemein kann aus der Verbindung dieses Betriebssystems mit der Kenntnis über die Bus-Signale und der Tastaturdekodierung gesagt werden, daß es eigentlich gar kein Atari 600XL/800XL-Betriebssystem gibt. Es handelt sich in jedem Fall um ein modifiziertes Betriebssystem von dem leider bei uns (noch ?) nicht ausgelieferten 1200XL !

Dies ist daran zu erkennen, daß das Betriebssystem des 600XL/800XL Tastenabfragen nach Funktionstaste 1 bis 3 enthält, diese Tasten jedoch nicht bei diesem Gerät vorgesehen sind. Der 1200XL dagegen besitzt vier Funktionstasten F1 bis F4. An diesen und wenigen anderen Details bleibt es dem Fachmann nicht unentdeckt, daß dieses Betriebssystem eigentlich kein 600XL/800XL-Programmpaket ist.

Im Unterschied zum alten 400/800-Betriebssystem sind einige Dinge erstaunlich und, um nach so viel positivem auch einmal negative Kritik anzubringen, merkwürdig.

So sind in dem System Routinen enthalten, die den völlig gleichen praktischen Wert haben wie die entsprechenden des alten 400/800-Betriebssystem, aber völlig anders aufgebaut sind und dabei nicht unbedingt besser, schneller oder einfacher. Auch sind einfach komplette Programmblöcke um nur wenige Byte verschoben worden, obwohl sie dort leicht hätten stehenbleiben können. Dies alles ist jedoch nur für diejenigen Programmierer ärgerlich, die sich nicht an die von der Firma Atari vorgegebenen Schnittstellen gehalten haben.

Es bleibt die Frage, ob der offensichtlich von Atari gewünschte 'Erziehungseffekt' überhaupt von den angesprochenen Personenkreisen bemerkt wird; wir glauben es (leider) nicht, da schon Mittel und Wege gefunden wurden, dem 600XL/800XL erfolgreich vorzutäuschen, er sei ein 400/800.

```
*****************************
*                           *
*                           *
*   RESET UND INTERRUPTS    *
* ------------------------- *
*                           *
*****************************
```

Beim Kaltstart, auch Power-Up-Reset genannt, werden sämtliche für den Betrieb benötigten Register und Hardwarebausteine des Atari initialisiert. Hierbei werden ebenfalls die Interrupts eingerichtet, die das System nach dem Kaltstart am Laufen halten.

Es gibt eine ganze Reihe unterschiedlicher Interruptsquellen im Atari 600XL/800XL. Als erste und wichtigste wäre der 50 Hz-Vertical-Blank-Interrupt zu nennen. Er gilt in diesem System als das Zeitnormal oder, anders formuliert, es wird nach ihm die (systeminterne) Uhr gestellt.

Beim Auftreten eines solchen Interrupts, der nur signalisieren soll, daß im Moment keine Bildinformationen gesendet werden, weil der Elektronenstrahl der Monitorbildröhre gerade von unten rechts nach oben links zurückwandert, werden nahezu alle im Hintergrund laufenden Prozesse bearbeitet, die nicht selbst eine Hardware-Unterbrechung auslösen können. Dazu gehören als wichtigste Prozesse die Verarbeitung der Schattenregister und der Timer.

Die Schattenregister haben, bei vom System zyklisch zu schreibenden Registern, die Aufgabe, die zu schreibenden Informationen zu sichern. Es hat bei vielen Registern keinen Sinn, vom Benutzerprogramm aus, einen Wert direkt in eben dieses Hardware-Register zu schreiben, da der entsprechende Baustein nach kurzer Zeit um ein Auffrischen der Information bittet. Ohne ein Schattenregister wüßte dann das System nicht, welchen Wert es übergeben soll. Hat man jedoch die gewünschte Information in ein Schattenregister, also eine ganz normale Speicherstelle geschrieben, kann das Betriebs-

system im Vertical Blank Interrupt den Inhalt dieser RAM-Zelle jedesmal auslesen und ihn in das Hardwareregister übertragen.

Die Verwendung von Schatten-Leseregistern hat meist eine andere Bewandtnis: Viele Meß- und Statusregister erkennen das Auslesen des gemessenen Wertes gleichzeitig als Rücksetzbefehl an, beginnen also mit einer neuen Messung oder löschen einfach nur bestimmte Informationen, wie zum Beispiel Fehlererkennungsbits.

Ist man nun gerade dabei, einen Wert auszumessen, und liest den bis dahin gemessenen Wert, ohne zu wissen, ob die Messung eigentlich abgeschlossen ist, würde in den meisten Fällen ein Fehlergebnis entstehen. Hat man im Gegensatz dazu eine einfache Speicherstelle, bei der man davon ausgeht, daß das Betriebssystem diese schon richtig beschreiben wird, ergeben sich diese Probleme nicht.

Ein weiterer Effekt der Schatten-Leseregister ist der, daß Meßergebnisse gezielt verändert werden können. So können von einem Statusregister nacheinander bestimmte Bits abgefragt werden. Sind sie gesetzt, wird ein dazugehöriger Handler aufgerufen. Bearbeitet dieser Handler nun nicht nur das eine Statusbit, sondern auch gleich noch einige andere, so kann dieser Handler nach der Behandlung die 'fertigen' Bits in der RAM-Zelle zurücksetzen, was ihm im Hardwarestatusregister nicht möglich gewesen wäre.

Die Timer haben die Aufgabe, zeitabhängige Vorgänge innerhalb des Systems zu koordinieren.

Es existieren zwei strikt voneinander zu trennende Arten von Timern: Die eine Sorte sind die Hardwarezähler, die im POKEY enthalten sind. Ist einer der Zähler 1, 2 oder 4 durch die angelegten Takte auf den Wert Null zurückgefallen, kann vom POKEY ein Interrupt ausgelöst werden. Auf diese Art Timer wollen wir hier nicht weiter eingehen, da ihre Funktionsweise eingehend im Kapitel POKEY erläutert ist.

An dieser Stelle interessieren eigentlich nur die fünf anderen Softwaretimer TIMCOUNT1 bis TIMCOUNT5.

Das sind jeweils 16bit-Werte, die vom Benutzer auf, in den gegebenen 16bit-Rahmen, beliebige Werte zu setzen sind und die dann selbsttätig bei jedem Vertical Blank Interrupt decrementiert werden.

Wird nach dem Verringern festgestellt, daß TIMCOUNT1 oder TIMCOUNT2 den Wert Null erreicht hat, wird indirekt über den entsprechenden Sprungvektor TIMER1VKT oder TIMER2VKT die vom Anwender zu programmierende Timer-Interruptroutine angesprungen. Diese Routinen sind immer mit einem normalen RTS bei 'aufgeräumtem' Stack zu beenden. Das heißt, daß nicht noch irgendwelche lokalen Werte auf dem Stack liegen dürfen, wenn zurückgesprungen werden soll.

Hat dagegen einer der Timer TIMCOUNT3, TIMCOUNT4 oder TIMCOUNT5 den Wert Null erreicht, so wird keine Routine angesprungen, sondern jeweils nur das entsprechde 8bit-Flag TIMER3SIG, TIMER4SIG oder TIMER5SIG vom Wert $00 auf $ff gesetzt. Das Anwenderprogramm mag dann die Veränderung dieser Werte selbst bemerken. Es ist auch nicht immer unbedingt sinnvoll, für jeden Timer-Event, also jeden Zeitpunkt, an dem eine bestimmte Operation ausgeführt werden soll, eine automatisch aufzurufende Prozedur zu schreiben, da es sein kann, daß der entsprechende Timer mehrfach herunterzählen soll, bis das gewünschte Ergebnis vorliegt.

Als weitere Operation wird im Vertical Blank Interrupt die sogenannte 'verzögerte' (deferred) Vertical Blank Interrupt-Routine aufgerufen. Da diese Routine im Normalfall nach einem Kaltstart nicht existiert, weist ihr Pointer VBLKDVKT auf die Interrupt-Ausgangsroutine EXITVBL. Diese Routine EXITVBL ist auch dann anzuspringen (nicht aufzurufen!), wenn eine solche verzögerte Unterbrechungsroutine existiert und beendet werden soll. EXITVBL stellt sicher, daß das ursprünglich unterbrochene Programm seine alten Registerwerte wiedererhält.

Wichtig zu erwähnen wäre, daß TIMCOUNT1 bei jedem Vertical Blank Interrupt bearbeitet wird, die restlichen Timer dagegen nur dann, wenn das Flag CRITICIO gelöscht ist. Dieses Flag soll verhindern, daß im vollen Betrieb bei sehr schnell folgenden Interrupts, also zum Beispiel beim Lesen von Floppy, der Rechner mit 'Kleinkram' wie Timerverarbeitung aufgehalten wird, und somit eventuell einen Interrupt nicht mehr rechtzeitig bearbeiten kann. Timer 1 wird deshalb immer benutzt, weil er die Aufgabe übertragen bekommen hat, eine mögliche Timeout-Situation zu erkennen. Dies kann er natürlich nur dann tun, wenn er regelmäßig bearbeitet wird.

Neben diesen regelmäßigen Unterbrechungen gibt es auch eine ganze Reihe asynchroner Interrupts, also solche, über deren zeitliches Auftreten nichts genaues gesagt werden kann. Dazu gehören zum Beispiel die Unterbrechungen der seriellen Übertragungsstrecke, die vom POKEY gesammelt und gewandelt werden. Diese und andere Interrupts besitzen ihre Vektoren ab der Adresse VPRECEDE.

Jede dort abgelegt Interruptroutine muß mit den Instruktionen

```
PLA
RTI
```

enden, um sauber zum unterbrochenen Progamm zurückzukehren.

```
**************************************
*                                    *
*                                    *
*  Ein-/Ausgabe über Kontrollblöcke  *
*  --------------------------------  *
*                                    *
**************************************
```

Beim Atari-Computer soll die gesamte Ein- beziehungsweise Ausgabe über sogenannte Kontrollblöcke ablaufen.

Das heißt, es gibt nicht für jedes anzusprechende Gerät eine eigene Adresse, die sich der Benutzer merken muß, sondern es gibt eine Adresse, die den Großteil des Input und Output übernimmt. Dazu wird die Beschreibung des Ein-/Asgabegerätes in einem Input-Output-Control-Block abgelegt. Dort stehen der Name des Gerätes, die Bus-Nummer, Pufferadressen und Pufferlängen sowie Statusinformationen:

IOCB - Eintrag	Bedeutung
IOCBCHID	IOCB-Channel-Identifikationsnummer. Sie ist der Offset des Eintrages innerhalb der Tabelle ab HATABS. Vorgegeben sind schon die Einträge

$00 =	'P'	Printer, Drucker
$03 =	'C'	Cassette
$06 =	'E'	Editor
$09 =	'S'	Screen, Bildschirm
$0c =	'K'	Keyboard, Tastatur
	'D'	Diskettenstation
	'M'	Modem

Die beiden letzten Einträge stehen nicht initiell in HATABS. Die Diskette nicht, weil sie nicht direkt über CIO sondern über eigene Disk-Operationen läuft, und das Modem deshalb nicht, weil es von Atari kein in Deutschland zugelassenes gibt. Die einzige Möglichkeit dazu wäre, über die Interfacebox zu arbeiten.

IOCBDSKN — Laufwerknummer. Die Diskette ist das einzige am Atari anzuschließende Peripheriegerät, von deren Sorte mehrere gleichzeitig am Bus anliegen können. Deshalb reicht die Information 'D' bei der Datenübertragung nicht aus und wird

um diese Laufwerknummer erweitert. Die Laufwerknummer kann theoretisch von 1 bis 9 laufen, es gibt jedoch nur Atari-Stationen, die die Nummern 1 bis 4 erkennen.

IOCBCMD — An dieser Adresse wird dem zentralen Ein-/Ausgabeprogrammpaket CIO das Kommando übergeben, was mit den Daten und dem angemeldetetn Gerät passieren soll.

Es stehen die folgenden, für alle Geräte geltenden, Kommandos zur Verfügung:

Befehlskode	Bedeutung
$03	OPEN Stellt fest, ob das angekündigte Gerät überhaupt am Bus anliegt.
$05	GET RECORD Lies den nächsten Block von dem angemeldeten Gerät, wenn es ein lesbares Gerät ist, sonst melde Fehler.
$07	GET CHARACTER(s) Lies das nächste Zeichen, beziehungsweise die nächsten Zeichen, bis zum NEWLINE von dem angemeldeten Gerät, wenn es ein lesbares Gerät ist, sonst melde Fehler.

$09 PUT RECORD
Schreibe den übergebenen Block an den geöffneten Datenkanal

$0b PUT CHARACTER(s)
Schreibe das nächste Zeichen, beziehungsweise alle folgenden Zeichen, bis zum nächsten NEWLINE an den geöffneten Kanal.

$0c CLOSE
Schließe den entsprechenden offenen Kanal und schreibe eventuell noch im Puffer vorhandene Daten vorher über den noch offenen Kanal.

$0d STATUS
Lies die Statusmeldung des angesprochenen Gerätes.

$0e SPECIAL
Rufe gerätespezifische Spezialroutine auf.

Weiterhin existieren einige Spezialkommandos, die jedoch nur bei jeweils einem Gerät gelten:

$11 DRAW LINE
Zeichnet eine Linie von einem Punkt zum anderen in dem programmierten Grafikmodus.

$12	DRAW LINE WITH RIGHT FILL Wie $11, nur daß rechts von der Linie bis zu einer eventuell rechts davon befindlichen weiteren Linie der Schirminhalt ausgefüllt wird.

Die beiden letzten Kommandos galten einsichtiger Weise nur für den Bildschirm, wobei die folgenden Spezialkommandos nur für den Diskettenzugriff gelten:

$20	RENAME DISK-FILE
$21	DELETE DISK-FILE
$22	FORMATIERE DISKETTE
$23	LOCK DISK-FILE Danach ist das angegebene File nur noch zum Lesen zu öffnen.
$24	UNLOCK LOCKED FILE
$25	POINT Dieses Kommando 'positioniert' den Schreib-/Lesekopf der Floppy-Disk logisch auf den angegebenen Sektor und das angegebene Byte.
$26	NOTE Analog zu POINT liefert dieses Kommando die Aussage darüber, wo sich die Diskettenstation gerade logisch befindet.

IOCBSTAT — In diesem Byte wird von CIOMAIN eine Meldung zurückgeliefert, an der das aufrufende Modul erkennen kann, wie die CIO-Routine das Kommando bearbeiten konnte. Es sind, abhängig von dem Gerät und dem Kommando, insgesamt folgende Statusmeldungen möglich:

$01	SUCCESS Die Operation verlief erfolgreich.
$80	BREAK_ABORT Währen der Kommandoausführung wurde die BREAK-Taste gedrückt.
$81	PREVIOUS_OPEN Das Gerät (beziehungsweise eigentlich sein Kanal) wurde schon einmal geöffnet und ist seitdem nicht wieder abgemeldet (geschlossen) worden.
$82	NON_EXISTANT_DEVICE Dieses Gerät ist in HATABS nicht bekannt.
$83	WRITE_ONLY Es wurde versucht, von einem Nur-Schreib-Kanal (zum Beispiel Drucker) zu lesen.
$84	INVALID_CMD Das übergebene Kommando ist nicht existent.

$85 NOT_OPEN
Das zu beschreibende oder zu lesende Gerät oder File ist noch nicht geöffnet.

$86 BAD_IOCBNR
Die übergebene IOCB-Nummer, die ein Vielfaches von 16 sein muß, ist entweder zu groß oder nicht durch 16 teilbar.

$87 READ_ONLY
Es wurde versucht, ein Nur-Lese-Gerät oder -File zu beschreiben.

$88 EOF_ERROR
Es wurde versucht, mehr aus einem File zu lesen, als darin enthalten ist.

$89 TRUNCATED
Bei einem Zeilen-Lese-Kommando wurde eine Zeile empfangen, die länger war als der zur Verfügung stehende Puffer. Die letzten Zeichen der Zeile sind nicht gespeichert, das letzte Zeichen des Puffers ist NEWLINE.

$8a TIMEOUT
Das Gerät hat sich nicht wieder gemeldet, nachdem es mit der Bearbeitung des Kommandos begonnen hat.

$8b DEVICE_NACK
Dieser negative Acknowe-ledge (negative Rückmel-dung) zeigt an, daß of-fensichtlich das anzu-sprechende Gerät nicht bereit (vorhanden, ange-schlossen, eingeschaltet o.ä.) ist.

$8c FRAMING_ERROR
Dieser Kode erscheint, wenn der POKEY beim Lesen eines Zeichens fest-stellt, daß das anzuhän-gende Stop-Bit nicht kam, sondern gleich mit der Übertragung eines weite-ren Zeichens begonnen wurde. Dieser Fehler tritt vor allem dann auf, wenn der Sender und Empfänger zu ungleiche Übertragungsraten besit-zen. Eine sinnvolle In-terpretation dieses Feh-lers kann jedoch auch die BREAK-Kondition sein.

$8d CURSOR_OVERRANGE
Der Cursor wird außerhalb des ihm zustehenden Bild-bereichs gesetzt. Tritt zum Beispiel bei falschen DRAWTO-Berechnungen auf.

$8e SIO-OVERRUN
Dieser POKEY-Fehler zeigt an, daß ein neues Zeichen eingelesen wurde, bevor das alte vom System gele-

sen worden ist. Damit ist das alte Zeichen verloren.

$8f SIO-CHECKSUM_ERROR
Die empfangene Prüfsumme stimmte nicht mit dem berechneten Wert überein.

$90 DEVICE_ERROR
Das Gerät konnte die Operation nicht vernünftig ausführen.

$91 BAD_SCREEN_MODE
Dieser Modus ist den bildschirmverarbeitenden Routinen unbekannt.

$92 FUNCTION_NOT_IMPLEMENTED
Die verlangte Operation ist zwar nicht verboten, aber dennoch nicht vorgesehen.

$93 INSUFFICIENT_SCREENMEMORY
Der zur Verfügung stehende Speicher reicht nicht aus, um den geforderten Bildschirmmodus zu initialisieren. Dies kann zum Beispiel passieren, wenn in einer 16KB-Maschine DOS geladen ist, ein BASIC-Programm läuft und eine hohe Grafikstufe gewählt werden soll.

Allgemein zu den Fehlermeldungen ist zu sagen, daß ihre Kodes bei entsprechender Abfrage das Negativ-Flag der CPU setzen.

IOCBBUFA — Hier wird die Anfangsadresse des (meist vom Benutzer zur Verfügung zu stellenden) Datenpuffers festgehalten. Der Wert ist unverändert nach Ausführung des Kommandos, unabhängig von Erfolg oder Mißerfolg der Operation.

IOCBPUTB — Hier steht die Anfangsadresse-1 der Routine, die an das definierte Gerät ein Zeichen überträgt. Der Vektor zeigt bei Nur-Lese-Geräten auf die entsprechende Fehlerbehandlungsroutine.

IOCBBUFL — Dies ist die Angabe der Länge des bei IOCBBUFA beginnenden Puffers.

IOCBAUXn — Die vier folgenden Byte sind Hilfsregister, von denen jedoch das erste, IOCBAUX1, zeitweilig für die CIO-Programmierung verwendet wird. Dabei sind folgende Werte erlaubt:

$01 APPEND
Dieser Kode erlaubt das anhängende Schreiben an eine bestehende Datei beziehungsweise das Lesen vom Bildschirm.

$02 DIRECTORY
Der folgende OPEN-Befehl veranlaßt einen Zugriff auf die Disketten-Directory.

$04 OPEN_FOR_INPUT
Dieses Kommando kann theoretisch bei jedem Gerät angewandt werden, es sind jedoch physikalische Einschränkungen zu

	beachten (zum Beispiel Drucker).
$08	OPEN_FOR_OUTPUT Es gilt das gleiche wie für OPEN_FOR_INPUT gesagte.
$12	OPEN_FOR_OUTPUT & INPUT Wie OPEN_FOR_INPUT.
$10	OPEN_FOR_MIXED_MODE Dieser Zusatz ist nur beim Editor und beim Screen erlaubt.
$20	OPEN_WITHOUT_CLEAR_SCREEN Ebenfalls nur für Editor und Screen erlaubt.

Um nun ein Kommando an die CIO-Routinen zu übergeben, wird im X-Register die Nummer des IOCBs * 16, und im Accu das unter Umständen zu sendende Zeichen an JUMPTAB+$06 übergeben.

Die Diskettenverarbeitung läuft nicht über die CIO-Routinen, da dort etwas andere Bedingungen vorliegen als bei den übrigen Geräten. Sie läuft über den Disc-Control-Block ab Adresse DSKDEVICE.

An diese und die folgenden Adressen werden folgende Werte übergeben:

DSKDEVICE — Bus-Kennummer der Diskette Nummer 1. Dieser Wert ist Referenz für die übrigen, nachfolgenden Geräte.

DSKUNIT — Hier steht die eigentliche Nummer des anzusprechenden Laufwerks.

DSKCMD	Es stehen die Kommandos '!' $21 FORMAT_DISKETTE 'P' $50 PUT_SECTOR_WITHOUT_VERIFY 'R' $52 GET_SECTOR 'S' $53 STATUS_REQUEST 'W' $57 PUT_SECTOR_WITH_VERIFY zur Verfügung.
DSKSTATUS	Hier steht nach der Bearbeitung eines Kommandos der Status der Operation.
DSKBUFFER	Anfangsadresse des Datenpuffers.
DSKTIMOUT	Anzahl der Sekunden bis zur Timeout-Meldung.
DSKBYTCNT	Länge des bei DSKBUFFER beginnenden Puffers.
DSKAUX1 DSKAUX2	Hilfsbyte. Hier steht bei den meisten Kommandos die zu lesende/schreibende Blocknummer.

Die Diskettenverarbeitung wird nicht über den CIO-Vektor angesprungen, sondern über JUMPTAB+$09, also das SIO-Interface.

```
*****************************
*                           *
*                           *
*  BETRIEBSSYSTEMROUTINEN   *
*  ----------------------   *
*                           *
*****************************
```

Im Folgenden werden die einzelnen Unterprogramme des Betriebssystems erläutert. Sie sind nach ihrer Lage (Adresse) im ROM geordnet. Die vier Zahlenangaben bedeuten von links nach rechts (die Adreßangabe kann sich beim 400/800 auf ungefähr gleichartige Routinen beziehen):

Adr. beim 600XL/800XL Hex	Dez	Adr. beim 400/800 Hex	Dez	Name
$c000	49152	$xxxx	xxxxx	CHECKSRO
$c001	49153	$xxxx	xxxxx	

Diese Adresse enthält die Prüfsumme über alle Bytes aus den Speicherbereichen

$c002 .. $cfff
$5000 .. $57ff
$d800 .. $dfff

Auf diesen Wert wird von CHECKROM1 zugegriffen.

$c00c	49164	$e6d5	59093	NMIENABLE

Es wird der NMI enabled und der Wert von TRIG3 in GINTLK gesichert. Beim 400/800 hat diese Routine zusätzlich die Aufgabe, die Portbausteine zu initialisieren.

$c018 49176 $e7b4 59316 NMIFIRST

Jeder NMI spingt indirekt über NMIVKT an diese Stelle. Hier wird getestet, ob es sich um eine ANTIC-Programm-Unterbrechung handelt. Wenn es eine ist, dann wird die ANTIC-Programm-Unterbrechung indirekt über DLIVKT angesprungen. Ist es keine ANTIC-Programm-Unterbrechung, wird nach dem Pushen des ACCU der RESET-Tastenstatus überprüft. Ist die RESET-Taste gedrückt, wird der Warmstartvektor (JUMPTAB + $24) angesprungen. Ist keine dieser Abfragen erfolgreich, so wird nach dem Retten der Register X und Y auf dem Stack indirekt VBLKIVKT aufgerufen.

$c02c 49196 $e6f3 59123 JMPIRQVKT

Jeder INT und jede BREAK-Operation laufen indirekt über INTVKT zu dieser Adresse. Hier wird im Gegensatz zum 400/800 das DECIMAL-Flag gelöscht, was gerade beim Erstellen von Interruptroutinen gerne vergessen wird und sonst beim 400/800 zu Fehlern führte. Da beim INT-Zyklus der 6202-CPU automatisch das Processor-Status-Word auf dem Stack abgelegt wird, hat das D-Flag nach einem RTI wieder den ursprünglichen Wert.

Nach Löschen des Flags wird die jeweilige Interruptroutine indirekt über VIMMEDIRQ angesprungen.

$c030 49200 $e70b 59147 SINRDYIRQ

Diese Routine übernimmt, angestoßen von einem Interrupt, die Kontrolle der seriellen und noch nicht existenter I/O-Einheiten. Dabei wird wie beim 400/800 getestet, ob ein Zeichen über den seriellen Port eingegeben worden ist. In diesem Fall müßte Bit 5 vom IRQST beziehungsweise IRQST§ gesetzt sein und wird die Einleseroutine indirekt über VSERIELIN angesprungen.

Weiterhin wird geprüft, ob die Verknüpfung (NEUIOMASK AND NEUIOPORT) einen Wert ungleich 0 ergibt, also ein an dieser Stelle angeschlossenes Gerät einen Interrupt erzeugt hat. Ist dies der Fall, so wird dessen Treiberroutine indirekt über NEUIOINIV angesprungen.

Ist auch diese Abfrage erfolglos, so wird, ähnlich wie beim 400/800, der IRQSTATUS dahingehend abgefragt, ob ein Interrupt des POKEY anliegt, der dem System mitteilt, daß das serielle Ausgaberegister zur Aufnahme eines neuen Datenbytes bereit ist, oder sogar die Übertragung des letzten Zeichens beendet ist. Die erste Bedingung wird durch Bit 4, die zweite durch Bit 3 des Statusbytes signalisiert. Ist Bit 4 gesetzt, so wird ein indirekter Ansprung zu der in VSERREADY stehenden Adresse vorgenommen, bei Bit 3 ein Ansprung zu der in VSERCLOSE stehenden Adresse. Da diese und die folgenden Abfragen in einer Schleife vorgenommen werden, wird der jeweilig gefundene Vektor (also zum Beispiel VSERREADY) nach NEUIOPTR geladen und von dieser, nun festen Adresse aus der eigentliche Treiber indirekt angesprungen.

Ist bis hierhin keine Interruptquelle gefunden, so kann es unter anderem auch der TIMER1, TIMER2 oder der TIMER4 des POKEY gewesen sein, der den Interrupt auslöste. Ist eines dieser Geräte aktiv gefunden, so wird der jeweilige Treiber (VTIMER1, VTIMER2, VTIMER4) nach der oben beschriebenen Methode aufgerufen.

Ist es auch kein Timerinterrupt, kann es noch eine Tastenanforderung gewesen sein. Hierbei werden schon an dieser Stelle zwei Möglichkeiten unterschieden: Jede normale Taste führt auf den Vektor VKEYBOARD, die erwähnte Ausnahme ist die BREAK-Taste. Ist sie gedrückt und KBDISABLE auf 0 (also enabled), wird indirekt nach VBREAKKEY gesprungen. Nur wenn bis hier keine Interruptquelle gefunden wurde, werden die IRQ-Bits des PIA abgefragt. Ist Bit 7 von PORTACNTL=1, so wird VPRECEDE aufgerufen, bei Bit 7 von PORTBCNTL=1 VINTERRUPT.

Hier kann nur noch ein softwaremäßiger BREAK anliegen, der im PSW der CPU signalisiert wird. Ist das BRK-Bit gesetzt, so wird die ab (VBREAK) liegende Routine aufgerufen.

Findet der Prozessor bis hier keine geeignete, aktive Interruptquelle, dann handelt es sich bei dieser Unterbrechungsanforderung um einen technischen "Irrtum" und die Routine kehrt zum unterbrochenen Programm zurück.

$c092 49298 $e785 59269 BRKEVENT

Diese Routine wird angesprungen, wenn die BREAK-Taste gedrückt wird und das Keyboard enabled ist (KEYDISABLE=0). Es werden hier der Attract-Mode, das STARTSTOP-Flag, CURSORINH und IRQST mit IRQST§ gelöscht. Damit ist das System unter normalen Bedingungen wieder arbeitsfähig.

Der Vektor VBREAKKEY zeigt auf BRKEVENT.

$c0cf 49359 $xxxx xxxxx MASKTAB

Diese Tabelle wird benutzt, um einzelne Bits ausblenden zu können. Sie enthält in aufsteigender Reihenfolge die Werte

$80, $40, $04, $02, $01, $08, $10, $20 .

$c0d7 49367 $xxxx xxxxx VECTAB

Diese Tabelle steht in direkter Verbindung mit MASKTAB bei der Verarbeitung der einzelnen Interruptquellen in SINRDYIRQ. Sie gibt zu der jeweiligen Maske den Offset des Ansprungvektors zu $200. Sie enthält in aufsteigender Rehenfolge die Werte

$36, $08, $14, $12, $10, $0e, $0c, $0a .

Bespiel: Es soll Bit 1 (also das 2. Bit!!!) getestet werden.

Also wird ein Offsetregister (X) mit 3 geladen und das entsprechende Statusbyte mit MASKTAB,X maskiert. Ist das Ergebnis gleich Null, so ist der Interrupt gefunden und es muß die Treiberadresse geholt werden. Da X immer noch den Wert 3 hat, wird einfach nach NEUIOPTR der Wert von $200,(VECTAB,X) geladen. (VECTAB,X) liefert den Wert $12, also wird der Vector aus der absoluten Adresse $212=$200+$12 geholt. Genau das gleiche geschieht mit NEUIOPTR+1 und $201,(VECTAB,X). Danach enthält NEUIOPTR den Pointer des Handlers des unterbrechenden Gerätes.

$c0df 49375 $xxxx xxxxx WAITFRRES

Dieser Programmteil sperrt sämtlich Interrupts und initiiert eine Warteschleife (65536 * Nichtstun), um danach RESET anzuspringen.

$c0f0 49392 $e7d1 59345 SYSTEMVBL

Diese Interruptroutine wird bei jedem Vertical-Blank-Interrupt aufgerufen und führt eine große Zahl von Systemkontrollen und Justierungen durch.

So wird als erstes die Atari-Uhr TIMER incrementiert. Diese Uhr ist eigentlich keine im üblichen Sinn, sondern sie zählt lediglich in 3 Byte (also TIMER, TIMER+1, TIMER+2) alle ankommenden Vertical Blank Unterbrechungen. Da bei unserem Fersehsystem alle 20ms (Milli-Sekunden: 1000ms=1s) ein sogenanntes Halbbild fertig sein muß, damit der Elektronenstrahl in der Bildröhre zur linken oberen Ecke zurücktransportiert werden kann, muß natürlich auch der Atari dafür sorgen, daß er zur selben Zeit die neuen Bildinformationen generiert. Unter anderem hierfür wird der Atari also alle 20ms unterbrochen, damit er die SYSTEMVBL-Routine

ausführen kann.

Dieses feste Zeitintervall kann nun auch für eine Uhr benutzt werden, mit dem kleinen Unterschied, daß sie nicht Sekunden, sondern 1/50 Sekunden angibt. Dabei ist TIMER+2 der niedrigste (also schnellste) Zähler, TIMER+1 ist der mittlere und TIMER zählt dann die Überläufe von TIMER+1. Damit ergibt sich auch gleich eine leichte Umrechnung von TIMER in die normale Zeit:

UHR = INT(((TIMER+2)+256*((TIMER+1)+256*(TIMER))) / 50)

UHR enthält dann die Anzahl der Sekunden ab Systemstart.

Bei jedem Incrementieren von TIMER+1 wird auch ATRACT erhöht. Dieses Register hat die Aufgabe, die Farben und Helligkeiten des Bildschirms zu verringern, wenn eine bestimmte Zeit lang kein Zeichen über die Tastatur eingegeben wurde. Hat ATRACT den Wert 128 erreicht (also nach einem Zeitraum von 128*256/50/60 = 10,9 Minuten), so werden Farbe und Helligkeit verändert. Dies geschieht dadurch, daß ATRACTMSK von $fe auf $f6 zurückgesetzt wird und COLREGSH mit dem Wert von TIMER+1 geladen wird. Diese Änderungen haben Einfluß auf die folgende Reinitialisierung der Farbe und Helligkeit von Vorder- und Hintergrund. Es gilt beim Beschreiben der Farb- und Luminanzregister folgende Anwendungsregel von ATRACTMSK und COLREGSH:

neue_Farbe+LUM = (alte_Farbe+LUM eor COLREGSH) and ATRACTMSK

Durch den Zusammenhang von TIMER+1 und COLREGSH entstehen die sich selbsttätig ändernden Farben auf dem Schirm im Attract-Mode.

Weiterhin werden an dieser Stelle die Schattenregister aufgefrischt, beziehungsweise neue Werte der Schattenregister in die Hardwareregister übertragen.

Im Einzelnen sind dies an dieser Stelle der Routine:

von	nach	Erklärung
LPENV	LPENV§	Lightpen vertikal
LPENH	LPENH§	Lightpen horizontal
DLPTR§	DLPTR	Display List Pointer
DMACNTL§	DMACNTL	DMA Control-Register
GTIACNTL§	GTIACNTL	Monitor Control-Register

FINESCROL wird, wenn es ungleich Null ist, decrementiert und der Wert ((8-FINESCROL) modulo 8) nach VSCROL geschrieben, um stetiges, weiches Scrollen zu ermöglichen.

Nun wird CONSOLE mit 8 geladen, um das Auslesen des Registers und somit das Erkennen eventuell gedrückter Sondertasten wie SELECT oder ähnlicher zu ermöglichen.

Dann wird die oben unter ATRACT erwähnte Reinitialisierung der Farb- und Luminanzregister durchgeführt. Es wird der Bereich COLPM0§ .. COLBAK§ in den Bereich COLPM0 .. COLBAK kopiert, wobei jeder Wert nach der oben angegebenen Gleichung modifiziert wird.

Dann werden CHARBASE mit CHARBASE§ und CHARCNTL mit CHARCNTL§ beschrieben. Der erste Wert liefert dem ANTIC die Highbyte-Adresse des gerade gültigen Zeichengenerators, von denen der Atari 600XL/800XL gleich zwei eingebaut hat, die jedoch auch über dieses Register um andere erweitert werden können. CHARBASE definiert, ob die Zeichen normal oder gedreht dargstellt werden sollen.

Als nächstes folgt ein Block zum Testen und Modifizieren der weiteren im System enthaltenen Timer TIMCOUNT2 bis TIMCOUNT5.

Diese 5 Timer sind jeweils 16-Bit Zähler, die alle 20ms über die Routine DECTIMER decrementiert werden. Den Timern TIMCOUNT0 und TIMCOUNT1 sind die Sprungvektoren TIMER1VKT sowie TIMER2VKT zugeordnet, die angesprungen werden, wenn nach dem Decrementieren die Zähler Null sind. Für die drei übrigen Zähler TIMCOUNT3, TIMCOUNT4 und TIMCOUNT5 werden beim Erreichen von Null die jeweiligen Flags TIMER3SIG bis TIMER5SIG gesetzt.

Der zeitliche Ablauf ist so, daß zuerst TIMCOUNT1 decrementiert und, bei Bedarf, TIMER1VKT aufgerufen wird. Dann erfolgen die weiter unten beschrieben Abfragen nur noch dann, wenn CRITICIO an dieser Programmstelle den Wert Null hat, also gelöscht ist.

Ist dies der Fall, wird TIMCOUNT2 decrementiert. Beim Erreichen von Null wird, über JMPTIMER2 und indirekt über TIMER2VKT, die entsprechende Routine aufgerufen.

Danach werden die Timer 3 bis 5 behandelt.

Als weiterer Block wird die Tastatur behandelt. Sie bietet über einige Register dem Anwender vielseitige Möglichkeiten zum Blockieren, Kodeändern oder Maskieren einzelner Tasten.

Die eigentliche Leseroutine testet zuerst, ob eine Taste gedrückt ist. Wenn keine gedrückt ist, prüft sie, ob der Delay-Zähler KEYDELAY Null ist. Ist er es nicht, wird er decrementiert, ansonsten wird mit der Bearbeitung der Joysticks fortgefahren.

Die Variable KEYDELAY verhindert, daß man zu schnell eingeben kann (Tastenentprellung), da der Rechner nur eine neue Tasteneingabe akzeptiert, wenn vorher die letzte Taste sozusagen 'abgeklungen' ist. Sie wird mit dem Wert 3 initialisiert.

Wenn eine neue Taste gedrückt wird, beginnt der, mit KREPDELY (Standardwert 40) initialisierte, Zähler SRTIMER rückwärts zu zählen, wenn diese entsprechende Taste gedrückt bleibt. Hat er bei einem Interrupt den Wert Null erreicht und ist diese Taste immer noch gedrückt und KBDISBLE=0, d.h. enabled, so beginnt ein Autorepeat der Tastatur. Um die Geschwindigkeit des Autorepeat festlegen zu können, wird ebenfalls SRTIMER benutzt. So lange, wie diese eine Taste gedrückt bleibt, wird SRTIMER mit dem Wert von KEYREP geladen und bei jedem Vertical-Blank-Interrupt rückwärts gezählt zu Null. Ist er Null, so wird der vom POKEY zur Verfügung gestellte Tastencode in Matrixform aus Register KBCODE, beziehungsweise seinem Schattenregister KBCODE§ ausgelesen.

KEYREP gibt somit die Repeat-Geschwindigkeit an und wird mit 5 initialisiert, während die in KREPDELY liegende Initialisierung von SRTIMER die Zeit angibt, bis die Repeatfunktion überhaupt einsetzt.

Da dieses Betriebssystem, wie oben schon erwähnt, eigentlich das 1200XL- Betriebssystem ist, geschehen an dieser Stelle nun einige für den 600XL/800XL unsinnige Dinge:

Es wird der gelesene Tastencode auf einige bestimmte Sonderzeichen hin überprüft. Der 1200XL verfügt zum Beispiel über vier freiprogrammierbare Funktionstasten, F1 bis F4. Merkwürdigerweise überprüft der Atari hier den Tastaturkode auf CNTL & F1, CNTL & F2, CNTL & F4, CNTL & 1 sowie einen offensichtlich in der Matrix der Tastatur nicht existenten Kode im Normalmodus, mit Shift und Control. Entspricht der gelesene Tastaturkode keinem der zwölf Werte, so wird dieser Kode in KBCODE§ abgelegt, sonst nicht. Interessant und kein Druckfehler ist, daß CNTL & F3 an dieser Stelle nicht geprüft wird.

Nach der Tastaturabfrage überprüft der Atari die Joystickports.

Diese Routine beginnt bei $c201, und kann angesprungen werden, wenn die vorhergehenden Abfragen alle nicht benötigt werden sollten. Es ist jedoch darauf zu achten, daß der normale Vertical Blank Interrupt in diese Routinenteile hineinläuft. Das eben gesagte gilt ebenso für die Triggerprüfung wie für den Paddle-Check.

Da der 600XL/800XL im Gegensatz zum 400/800 nur noch über zwei Joystickports verfügt, einige wenige Software jedoch auf Port 3 und/oder Port 4 zugreift, hat sich Atari aus Kompatibilitätsgründen dazu entschlossen, die fehlenden Ports zu simulieren.

Es werden zuerst die oberen 4 Bit des PORT A des PIA gelesen und gleichzeitig als Wert für JOYSTICK1 und JOYSTICK3 verwertet; beim 400/800 wurde der Wert für JOYSTICK3 aus PORT B 'gezogen'. Danach werden die unteren vier Bit von PORT A nach JOYSTICK0 und JOYSTICK2 transportiert.

Nun werden (ab $c219) die Triggereingänge überprüft. Auch hier werden die Daten von Port 1 und Port 2 ebenfalls für jeweils Port 3 und Port 4 in der Form verwertet, daß sie aus TRIGGER0 und TRIGGER1 gelesen und nach TRIGGER0§ und TRIGGER2§, beziehungsweise TRIGGER1§ und TRIGGER3§ geschrieben werden.

Ähnliches gilt für das bei $c22b beginnende Lesen der Paddle-Eingänge. Die POKEY-Register PADDLE0 .. PADDLE3 werden nach PADDLE0§ .. PADDLE3§, beziehungsweise nach PADDLE4§ .. PADDLE7§ kopiert und die Paddle-Trigger gelesen. Dabei gilt folgende Zuordnung:

```
Aus JOYSTICK0   bit  2  ==:   PTRIG0, PTRIG4
    JOYSTICK0   bit  3  ==:   PTRIG1, PTRIG5
    JOYSTICK1   bit  2  ==:   PTRIG2, PTRIG6
    JOYSTICK1   bit  3  ==:   PTRIG3, PTRIG7
```

An dieser Stelle ist die erste Vertikal-Blank-Unterbrechungsroutine abgeschlossen und es folgt ein indirekter Sprung über VBLKDVKT zur sogenannten 'verzögerten' V-Blankroutine. Sie ist vom Benutzer frei zu verwenden. Es ist allgemein bei der Verwendung dieses Vektors darauf zu achten, daß die Interruptroutine nicht zu lang wird, da sonst der eigentliche Rechenprozeß (zum Beispiel BASIC) unter Umständen nicht mehr zum Zug kommt.

$c25d 49757 $e8ef 59631 JMPTIMER1

Es wird ein indirekter Sprung mit VTIMER1 durchgeführt.

$c260 49760 $e8f2 59634 JMPTIMER2

Es wird ein indirekter Sprung mit VTIMER2 durchgeführt.

$c263 49763 $e8f5 59637 DECTIMER

Die Routine DECTIMER decrementiert einen angegebenen Timer, wenn er nicht Null ist. Ist der betreffende Timer entweder vor dem Decrement Null oder hinterher ungleich Null, so liefert DECTIMER im Accu den Wert $ff zurück, sonst den Wert $00. Durch das Laden kann das aufrufende Programm den Wert des Zero-Flags verwerten. Ist Z=1, so ist der Timer gerade eben zu Null geworden, sonst ist Z=0.

DECTIMER erwartet die Nummer des zu behandelnden Timers, multipliziert mit 2, im X-Register beim Aufruf; also zum Beispiel in X den Wert $06 für TIMCOUNT3.

$c280 49792 $e912 59666 SETVBLVKT

Diese Routine wird allgemein verwendet, um Interrupt-

vektoren oder Timerwerte zu initialisieren. Dazu müssen an SETVBLVKT folgende Werte übergeben werden:

Wert	Übergeben in Register
High-Byte der zu speichernden Adresse beziehungsweise des zu speichernden Wertes	X
Low -Byte der zu speichernden Adresse beziehungsweise des zu speichernden Wertes	Y
Registernummer	A

Vektornummer kann sein:		
	0 ==:	VIMMEDIRQ
	1 ==:	TIMCOUNT1
	2 ==:	TIMCOUNT2
	3 ==:	TIMCOUNT3
	4 ==:	TIMCOUNT4
	5 ==:	TIMCOUNT5
	6 ==:	VBLKIVKT
	7 ==:	VBLKDVKT

Die Verktornummer kann auch größer als 7 (oder unsigned 7 bit) werden, dann ist jedoch darauf zu achten, daß die Routine nur 16bit-Worte und die nur ab geraden Adressen ablegen kann.

Vor dem Modifizieren der Vektoren wird ein Vertical Blank Interrupt abgewartet, damit die Vektoren nicht gerade 'halb geändert' sind, wenn ein Interrupt auf sie zugreifen will.

$c298 49816 $e93e 59710 EXITVBL

Diese Routine macht nichts weiter, als Y-Register, X-Register und Accu in dieser Reihenfolge vom Stack zu

holen und mit RTI diesen (beliebigen) Interrupt zu beenden.

$c29e 49822 f11b 61723 RESETWARM

RESETWARM kann direkt oder über JUMPTAB+$24 aufgerufen werden. Zu Beginn der Warmstartroutine wird getestet, ob nicht vielleicht doch gravierende Einflüsse auf das System eingewirkt haben, die einen Kaltstart mit Initialisierung sämtlicher Register nötig machen würden. Zu diesen Einflüssen gehört zum Beispiel das Herausziehen oder Wiedereinstecken eines ROM-Moduls in den ROM-Schacht. Da beim 600XL/800XL im Gegensatz zum 400/800 nicht mehr automatisch das Gerät beim Wechseln eines ROM-Moduls ausgeschaltet wird, ist hier also solch ein Test notwendig.

Dazu wird der Inhalt von TRIGGER3 des POKEY gelesen und mit dem Inhalt des für diesen Zweck eingerichteten ROMFLAG verglichen. Welchen Zweck das Lesen des TRIGGER3 hat, verrät uns ein Blick in das offene Gerät: Die Leitung, die beim Einstecken des ROM-Modules das eventuell an gleicher Stelle liegende RAM abschaltet, ist mit eben dieser Triggerleitung verbunden, die wegen Weglassens der Joystickports 3 und 4 frei geworden ist. Wenn nun seit dem letzten Reset entweder ein ROM eingesteckt oder entfernt worden ist, stimmen TRIGGER3 und ROMFLAG nicht überein und der Atari forciert einen RESETCOLD-Aufruf.

Sind TRIGGER3 und ROMFLAG noch identisch, so kann es zum Beispiel sein, daß zwar wirklich ein ROM-Modul im Schacht steckt, es jedoch ein anderes als beim letzten Reset ist. Für diesen Fall werden die letzten Adressen des ROMs getestet, genauer: Durch Aufruf von NEWCART wird die Checksumme von $bff0 bis $c0ef gebildet und mit der Checksumme des letzten Aufrufs verglichen. Ist die Checksumme nicht in Ordnung, so wird ein Kaltstart durchgeführt.

Es gibt noch eine dritte Möglichkeit, bei der ein Warmstart zu einem Kaltstart werden kann: Durch Setzen von COLDSTART auf einen Wert ungleich Null, wird dieses erreicht.

Hat keine der genannten Quellen bis hierhin einen Kaltstart forciert, wird RESET+$02 mit einem Accu-Wert von $ff aufgerufen, das heißt, die eigentliche Warmstartprozedur durchgeführt.

$c2b8   49848 $f125   61733   RESETCOLD

Zu dieser Routine kann direkt oder über JUMPTAB+$27 gesprungen werden. Der Weg über JUMPTAB ist sinnvoller, da dann Programmkompatibilität zu der 400/800-Serie besteht.

Zu Beginn des Kaltstarts wird, analog zu RESETWARM, geprüft, ob es sich wirklich um einen solchen handelt, oder aber seit dem letzten Kalt- oder Warmstart keine gravierenden Einflüsse auf das System eingewirkt haben, so daß der Rechner mit einem Warmstart auskommen könnte.

Dazu wird als erstes geprüft, ob drei Adressen, die beim Kaltstart auf ganz bestimmte Werte gesetzt werden, diese immer noch enthalten. Die Adressen und ihre Inhalte sind:

POWUPBYT1	muß für Warmstart enhalten	$5c
POWUPBYT2	muß für Warmstart enhalten	$93
POWUPBYT3	muß für Warmstart enhalten	$25

Was diese Werte bedeuten, ist recht unklar, es wird jedoch davon ausgegangen, daß ein neu eingeschalteter Speicherbaustein (RAM) bestimmt nicht diese drei Werte an eben diesen Adressen enthalten kann. Normalerweise sind dynamische Speicherzellen nach dem Power-up bankweise entweder $00 oder $FF, bedingt durch ihren physikalischen Aufbau.

Wenn eine dieser drei Adressen nicht den Warmstartwert enthält, wird WARMFLAG mit $00 geladen, was der danach aufgerufenen Routine RESET mitteilt, daß es sich um einen Kaltstart handelt.

Sind die drei Adressen in Ordnung, wird RESETWARM aufgerufen.

$c2d6	49878	$f126	61734	RESET
$c2d8				

In diesem Programmteil werden, je nachdem, ob es sich um einen Kalt- oder Warmstart handelt, mehr oder weniger Register und Bausteine initialisiert. Die eigentliche Entscheidung darüber, ob es sich um einen Warm- oder Kaltstart handelt, wird nicht mehr hier getroffen, sondern in RESETWARM beziehungsweise RESETCOLD. Wird RESET aufgerufen, wird das WARMFLAG auf $00 gesetzt, was der weiteren Routine einen Kaltstart signalisiert. Wird RESET+$02 aufgerufen, so wird in WARMFLAG der Wert des übergebenen Accus abgelegt. Ist dieser Wert = $ff, so signalisiert WARMFLAG einen Warmstart.

Danach beginnen die Resetoperationen. Zuerst wird DOSVKT auf den Wert TESTROMEN gesetzt. Wird beim Einschalten der Maschine oder einem anderen Kaltstart die OPTION-Taste gedrückt, wird das im 600XL/800XL befindliche BASIC ausgeschaltet. Ist keine Diskette angeschlossen oder antwortet sie nur nicht richtig, wird der Atari nach einiger Zeit versuchen, etwas anderes sinnvolles zu tun als zu booten. Da er jedoch kein BASIC zur Verfügung hat und beim 600XL/800XL kein Monitor mehr besteht (der 400/800 ging in einen Echo-Modus, wenn er weder ROM noch Bootgerät fand), wird er diesen DOSVKT anspringen, um nicht abzustürzen. Über diesen Vektor wird er dann bei TESTROMEN das eingebaute Test-ROM einschalten und anspringen.

Bevor er jedoch dies alles tun kann, hat er noch eine ganze Reihe anderer Dinge zu tun.

Zuerst wird über CARTGO getestet, ob im Schacht ein ROM-Modul steckt. Ist dies der Fall, und ist dieses ROM ausführbar (siehe CARTGO), so kehrt diese Routine nicht zurück, sondern springt das ROM indirekt über $bffe an.

Als nächstes wird über NEWCART die Checksumme über die letzten Bytes des eventuellen ROMs gebildet und in ROMFLAG abgelegt.

Weiter wird dort geprüft, ob das BASIC einzuschalten ist und die Ports werden initialisiert.

Danach beginnt der Atari mit dem RAM-Test, falls es sich um einen Kaltstart handelt. Dabei werden die entsprechenden Variablen von SYSINIT gesetzt.

Ist dies geschehen, oder handelt es sich um einen Warmstart, werden die Variablenbereiche $200 .. $3ed und $00 .. $7f mit $00 initialisiert. Nun wird X64KBFLAG auf den Wert gesetzt, den Bit 1 von PORT B besitzt und die Variablen POWUPBYT1, -2 und -3 werden mit den Werten $5c, $93 und $25 geladen. Weiter werden der rechte und linke Rand des Textfensters in RIGMARGIN und LFTMARGIN eingestellt auf die Werte 39 beziehungsweise 2.

Als nächstes kommt eine Eigenschaft des 600XL/800XL - Betriebssystems zu Tage, über die der 400/800 nicht verfügt. In den USA ist bekanntermaßen ein anderes Farbsystem verfügbar als bei uns. Abgesehen von unterschiedlicher Bildqualität ist die Signalzusammensetzung und die Bildfrequenz nicht mit 50Hz (Hertz, 1Hz = 1 komplette Schwingung pro Sekunde), sondern mit 60Hz vorgegeben. Da jedoch aus dieser Freuenz über den Vertical Blank Interrupt ein Zeitnormal gezogen wird, ist dieser Unterschied wichtig zu beachten. Bei den alten Systemen wurde das so gehandhabt, daß einfach überall die Zeitkonstanten verändert wurden, die auf den Zeittakt zurückgriffen. Da aber den Entwicklern der neuen Geräte aufgegangen ist, daß 'Old-Germany' inzwischen auch über einen recht großen Computermarkt verfügt,

haben sie die Zeitkonstanten parametrisiert, das heißt, die Zeitkonstanten sind alle über Tabellen zu erreichen.

Um nun jedoch zu unterscheiden, ob es sich bei diesem Gerät um unser PAL- oder das NTSC-System handelt, wird ein Register des GTIA gelesen, und zwar NTSCPAL. Sind die Bits 1 bis 3 dieses Registers alle Null, so handelt es sich um einen PAL-Baustein, sonst um einen NTSC-Typ. Je nach Farbsystem werden nun die folgenden Register unterschiedlich geladen:

Register	NTSC	PAL	Erklärung
NTSCPAL§	$00	$01	Flag zur weiteren Verwendung
KEYRPDELY	$30	$28	Initialisierungswert für SRTIMER, wenn Taste neu gedrückt worden ist
KEYREP	$06	$05	Initialisierungswert für SRTIMER, wenn Taste gedrückt ist und SRTIMER wenigstens einmal abgelaufen ist.

Um die gesamte Interrupt-, Timer- und Vektormaschinerie zum Laufen zu bringen werden die Adressen $200 .. $225 (DLIVKT bis VBLKDVKT) initialisiert mit den Werten, die ab Adresse INIT200 stehen. Direkt danach werden die Handlervektoren ab HATABS mit den vorgegebenen Adressen ab INIT31A initialisiert.

Aus den Speichertestroutinen ist noch ein Flag zu verarbeiten: Als Hilfsregister wurde Adresse $0001 mit dem Wert $00 oder einem anderen für das Ergebnis des Speichertests verwendet.

Ist das Ergebnis nicht Null, so wird über GOMEMTEST das Test-ROM eingeschaltet, die Variablen CHARCNTL mit $02 und CHARBASE§ mit $e0 (für Zeichengenerator ab $e000) initialisiert und direkt der Speichertest aufgerufen.

Hat $0001 den Wert Null, wird IOCB0 bestückt. Es werden das Kommando OPEN, die Namenspufferadresse OPENSCST sowie das IOCBAUX1 mit READ + WRITE (s. Erklärung IOCB-Verwendung) progammiert. In Worten heißt das also, daß IOCB0 für einen Schreib/Lese-Open für den Screen-Editor vorbereitet wird. Dann wird versucht, über CIOMAIN diesen Open auszuführen. Da es dabei eigentlich keine Probleme geben dürfte, wird bei einem eventuellen Fehler unmittelbar ohne zweiten Versuch RESETCOLD ausgeführt.

Ist bis hierhin alles glatt gegangen, wird geprüft, ob von Kassette gebootet werden soll (Drücken der START-Taste), oder eine bootbare Diskette existiert. Wenn ja, wird über BOOT geladen, sonst wird, falls TMPRAMSIZ = $01 ist und bit 2 von Adresse $bffd (im ROM) gesetzt ist, indirekt über COLDCART das ROM angesprungen.

Ist auch dieses nicht der Fall, wird der inzwischen eventuell geänderte Vektor DOSVKT indirekt als Sprungvektor benutzt.

$c3bd 50109 $xxxx xxxxx GOMEMTEST

Hier wird Bit 0 von PORT B auf 0 gesetzt. Damit wird das im Bereich von $5000 bis $57ff befindliche RAM ausgeschaltet und das im Bereich $d000 bis $d7ff hinter den I/O-Bausteinen versteckt liegende Test-ROM wird über die Memory Management Unit (MMU) auf den Bereich $5000 bis $57ff kopiert. Dieses scheinbar schwierige Kopieren wird einfach dadurch erreicht, daß die MMU unter diesen Bedingungen bei einer Adresse %y101 xxxx xxxx xxxx (x=egal) das y-Bit auf 1 setzt. Mit anderen Worten addiert die MMU im Falle des Ansprechens einer Adresse $5xxx den Wert $8000 und erhält $dxxx.

Hier sitzt nun also das Test-ROM, das (wenigstens) 2 Einsprünge enthält: Die Adressen TESTROM und TESTROM+$03. Über TESTROM gelangt man in das Menue, das man auch beim Kaltstart mit gedrückter OPTION-Taste erhält. TESTROM+$03 geht direkt zum Memory-Test.

Nach dem Umschalten wird nun also einfach TESTROM+$03 angesprungen.

$c431 50225 $xxxx xxxxx COLDCARTC

Es wird ein indirekter Sprung nach (COLDCART) unternommen.

$c434 50228 $xxxx xxxxx DOSVKTC

Es wird ein indirekter Sprung nach (DOSVKT) unternommen.

$c437 50231 $xxxx xxxxx INITCARTC

Es wird ein indirekter Sprung nach (INITCART) unternommen.

$c43a 50234 $xxxx xxxxx CLCUNDRTS

Wie der Name der Routine schon vermuten läßt, wird hier nur das Carry-Flag gelöscht und Return ausgeführt. Diese (scheinbar unsinnige) Routine kann leicht auch von Anwenderprogrammen zum Signalisieren von OK-Meldungen bei Prozedurausgängen benutzt werden.

$c43c 50236 $f0e3 61667 INIT31A

Diese Tabelle enthält die Werte, mit denen die Handler-Ansprungtabelle ab HATABS initialisiert wird.

$c44b 50251 $f10d 61709 DERRMSG

Hier liegt die Meldung "BOOT ERROR (CR)"

$c459 50265 $e480 58496 INIT200

An dieser Stelle liegen die Daten, die von RESET in die Interrupt- und Timervektoren, beziehungsweise Timer selbst, geladen werden.

$c47f 50303 $xxxx xxxxx CARTGO

Diese Routine testet, ob ein Cartridge gesteckt ist. Wenn diese Bedingung erfüllt ist und die im ROM an Adresse $bffc und $bffd stehende Adresse größer oder gleich dem Wert $8000 ist, wobei es eine $x000 - Adresse sein muß, so wird ein Sprung indirekt über $bffe ausgeführt.

Wenn dies nicht der Fall ist, werden über IOPORTINI die verschiedenen Ports initialisiert und geprüft, was mit Bit 1 von PORT B zu geschehen hat. Dieses Bit ist ein Ausgang und gibt an, ob das eingebaute BASIC aktiv werden kann oder nicht. Das BASIC ist eingeschaltet, wenn Bit 1 des PORT B den Wert Null hat. Es bekommt diesen Wert, wenn X64KBFLAG Null ist und die OPTION-Taste nicht gedrückt ist. Letzteres erfährt man über Bit 2 von CONSOLE: Ist das Bit gesetzt, so ist die Taste nicht gedrückt und umgekehrt. Nach Setzen, beziehungsweise Rücksetzen, dieses Bits ist die Routine beendet. Da sie jedoch keine eigenständige Prozedur ist, läuft sie in die nächstfolgende Speichertestrou-

tine GETRAMHI hinein und kehrt von dort aus zum Aufrufer zurück.

$c4b7 50359 $f258 62040 GETRAMHI

Diese Routine liefert in Register TMPRAMSIZ die Anzahl der, dem Benutzer zur Verfügung stehenden, vollen 256-Byte-Blöcke an RAM (Programmspeicher). Dabei wird, bei Adresse $2800 mit dem Test beginnend, jedes erste Byte eines solchen Blockes gelesen und das Einerkomplement dieses Wertes an eben diese Stelle zurückgeschrieben. Stimmt danach der Speicherzelleninhalt nicht mit dem 'gemerkten' Einerkomplement überein, so handelt es sich hierbei offensichtlich nicht um eine RAM-Zelle und das Programm terminiert. Stimmen die beiden Werte überein, wird der Originalwert wieder zurückgeschrieben und wieder gelesen.

Ließ sich die Speicherstelle wieder umprogrammieren, so handelt es sich hierbei mit ziemlicher Sicherheit um eine RAM-Zelle, ansonsten wird das Programm ebenfalls abgebrochen. Da zum Testen die Adressen TMPRAMSIZ-$01 und TMPRAMSIZ als Pointer auf die zu testende Speicherstelle dienen, liefert also TMPRAMSIZ als High-Wert des Pointers das High-Byte der ersten nicht-RAM-Adresse und somit die Anzahl der zur Verfügung stehenden, darunterliegenden Blöcke.

$c4d7 50391 $xxxx xxxxx NEWCART

Hier wird die Checksumme über alle Bytes von $bff0 bis $c0ef gebildet und mit CARTCKSUM verglichen. Sind die beiden Werte gleich, kehrt die CPU mit gesetztem Zero-Flag zurück, ansonsten wird der neue Wert in CARTCKSUM abgelegt und das Zero-Flag vor dem Rücksprung gelöscht.

$c4e8 50406 $f281 62081 IOPORTINI

In diesem Programmteil werden alle bekannten I/O-Ports und Bausteine initialisiert. Genauer werden die Bereiche

$d000 ... $d0ff (GTIA)
$d200 ... $d2ff (POKEY)
$d300 (PORT)
$d302 ... $d3ff (PORT)
$d400 ... $d4ff (ANTIC)

gesamt auf $00 gesetzt. Danach wird PORT B auf Ausgang geschaltet und alle Bits dieses Ports auf 1 gesetzt. Danach werden die Statusworte von PORT A und B gelesen, um eventuell anliegende Interrupts zu löschen.

Sind die Ports initialisiert, wird der POKEY dahingehend gesetzt, daß er seine Sende- und Empfängertakte von Audiokanal 4 erhält. Weiter werden die Audiokanalregister AUDCNTL3 und -4 auf den Wert $a0 und AUDIOCNTL auf den Wert $28 gesetzt. Abschließend wird ein $ff - Byte an das serielle Senderegister geschickt.

$c543 50499 $f294 62100 SYSINIT

Diese von RESET aufgerufene Prozedur hat die hauptsächliche Aufgabe, sämtliche I/O-Bausteine zu initialisieren. Dazu setzt sie als erstes nach Löschen des BREAK-Enable-Bits in IRQST§ den BRKEYVKT auf BRKEVENT ($c092). Danach wird RAMSIZE mit dem Wert von TMPRAMSIZ geladen und MEMTOP ebenfalls entsprechend gesetzt, um danach die Untergrenze des Benutzer-RAMs in MEMLO mit dem Wert $700 festzulegen.

Danach werden in dieser Reihenfolge die Initialisierungsroutinen des Editors, des Screens, des Keyboards, des Printers und der Kassette über die entsprechenden Vektoren ab EDITORVKT aufgerufen.

Hiernach fehlen noch die ersten Aufrufe von CIOINIT, SIOINIT, und NMIENABLE um die Interruptstruktur des Systems komplett zu nutzen sowie ein Aufruf von DISKINIT.

Kehrt die CPU nach diesen Routinen zurück, so wird NEUIOINIV auf NEUIOREQ ($c97c) gesetzt und über NEUINITC endlich NEUINIT aufgerufen.

Nach diesen Hardwarebearbeitungen wird vor dem Rückkehren aus der Routine noch STARTTST auf $01 gesetzt, wenn die START-Taste gedrückt ist, ansonst wird STARTTST zu $00. Dieser Wert wird später von BOOT benutzt.

$c599   50585 $f2cf   62159   BOOT

Zu Beginn dieser Boot-Routine wird geprüft, ob es sich um einen Boot-Ansprung bei einem Warmstart handelt. Ist dies der Fall (WARMFLAG ungleich $00) und außerdem ein vernünftiges Disk Operating System geladen (DOSAKTIV = 1), so wird ein indirekter Sprung nach (DOSINITV) vollzogen.

Ist es zwar ein Warmstart, aber ist kein DOS geladen, so wird einfach die Bootroutinenbearbeitung abgebrochen.

Der letzte Fall ist der eigentlich interessante in dieser Prozedur:

Zuerst wird versucht, von der Diskettenstation Nummer 1 eine Statusmeldung zu erhalten. Dies geschieht nach Setzen des STATUSCMDs in DSKCMD, Setzen der Nummer 1 in DSKUNIT und Aufruf von DISKINTERF über JUMPTAB+$03. Kehrt diese Routine mit gelöschtem Negativ-Bit in der CPU zurück, so war der Status in Ordnung und es kann von Diskette gebootet werden. Wenn nicht, kehrt BOOT mit gesetztem Negativ-Bit zurück.

Ist die Statusabfrage positiv verlaufen, wird Sektor Nummer 1 ab Adresse $0400 geladen. Dazu wird die Blocknummer $0001 nach DSKAUX1 (Low-Byte)und DSKAUX2 (High-Byte) geladen sowie der Wert $0400 nach DSKBUFFER, um danach den Block mit GETSECTOR zu lesen. GETSECTOR kann sowohl von Kassette als auch von Diskette lesen und liefert im Falle eines falsch oder nicht gelesenen Blockes einen negativen Wert zurück.

In diesem Fall wird die Meldung "BOOT ERROR" ausgegeben und beim Laden von Cassette das Booten abgebrochen, ansonst ein erneuter Leseversuch gestartet.

Vor der Beschreibung der weiteren Vorgänge empfiehlt es sich, zwei artverwandte Begriffe zu klären:

Ein SEKTOR ist ein auf der Diskette eingetragener Satz von Daten. Beim Atari existieren pro Diskette 40 konzentrische Spuren, die jeweils 18 Sektoren mit jeweils 128 Byte Daten enthalten. Die Tracks werden innerhalb der Floppy-Disk-Station von $00 bis $27 durchnumeriert, die Sektoren auf jedem Sektor verwirrender Weise von $01 bis $12. Warum die unterschiedliche Zählweise eingeführt worden ist, ist unklar, sie wurde jedoch nicht von Atari festgelegt, sondern von dem Hersteller des Floppy-Disk-Controllers innerhalb der Diskettenstation.

Im Gegensatz zum physikalischen Sektor ist ein BLOCK eine logische Zusammenfassung von Daten. Sie kann theoretisch beliebig viele Datenbytes enthalten. Der Einfachheit halber enthält ein Block beim Atari ebenso viele Datenbytes wie ein Sektor, er wird nur nicht pro Track gezählt, sondern durchgehend von $01 bis $02d0. Für den Atari-Anwender ist normalerweise nur die Betrachtung der Blöcke von Interesse, da die Diskettenstation selbst eigene Intelligenz besitzt und somit dem Benutzer keine Gelegenheit gibt, an die Unterscheidung von Sektoren und Tracks heranzukommen. Trotzdem sollte der Zusammenhang einmal geklärt werden, da bei der Benutzung von Erweiterungen an den Atari (CP/M-Systemen

zum Beispiel) diese Trennung einmal interessant werden kann. Spätestens bei der Verwendung von Diskettenstationen mit doppelter Schreibdichte enthält ein Block nur halb so viele Daten wie ein Sektor, so daß das Betriebssystem hier einige Daten-"Schaufelarbeit" zu erledigen hat. Aber dazu später mehr.

Ist der erste Block richtig gelesen, werden die ersten vier Byte nach DSKFLAG, DSKSECCNT und DSKLDADR kopiert. Sie geben die Sektorlänge, die Anzahl der zu lesenden Blöcke und die Startadresse, ab der die gelesenen Daten abgelegt werden sollen, an. Die nächsten zwei Byte geben die Startadresse des Bootfiles an und werden in DOSINITV abgelegt.

$c5c9 50633 $f301 62209 BLOCK1

(Dieses Label steht hier deshalb recht verloren inmitten der Routine, weil, von CASBOOT aus, hier hineingesprungen wird!)

Danach wird der gesamte gelesene Block ab der angegebenen Bootadresse abgelegt, DSKSECCNT decrementiert und, falls noch ein Block zu lesen ist, dieser an den jeweils folgenden Adressen abgelegt. Tritt ein Lesefehler auf, so wird beim Booten von Cassette sofort die Bearbeitung abgebrochen, beim Diskettenboot wird versucht, das gesamte Bootfile ein weiteres Mal einzulesen.

Sind alle Blöcke richtig eingelesen und wurde von Cassette gebootet, so wird noch ein END_OF_FILE-Block eingelesen, der jedoch nicht weiter verwendet wird.

Ansonst wird über BLOAD der Anfang der Bootroutine aufgerufen. Dieser Anfang liegt im sechsten Byte des ersten gelesenen Blocks.

Bei einem Diskettenprogramm kann hier getestet werden, ob wirklich alles richtig geladen worden ist.

Wenn alles in Ordnung ist, muß dieser Test mit gelöschtem Carry-Flag zurückkehren, damit nicht noch einmal versucht wird, zu booten.

Ist das Booten erfolgreich verlaufen, wird über DOSINITC das Diskettenbetriebssystem initialisiert und danach DOSAKTIV auf 1 gesetzt.

$c637 50743 $f36c 62316 BLOAD

Es wird der Wert von DSKLDADR+6 nach RAMTSTPTR geladen und danach indirekt über RAMTSTPTR die gebootete Routine angesprungen.

$c649 50761 $f37e 62334 DOSINITC

Es wird ein indirekter Sprung über DOSINITV ausgeführt.

$c64c 50764 $f381 62337 DSKRDERR

die CPU-Register X und Y werden mit der Low-, beziehungsweise High-Adresse der Fehlermeldung DERR geladen. Dieser Programmteil kehrt nicht alleine zurück, sondern läuft in die folgende Routine hinein.

$c650 50768 $f385 62341 PUTLINE

Diese Routine erwartet im X-Register der CPU den Low- und im Y-Register den High-Wert der Adresse, ab der ein auszudruckender Text im Speicher steht. Die Textzeile muß mit dem Atari-spezifischen NEWLINE($9b) enden und darf nicht länger als 255 Zeichen sein. Die Routine zerstört eventuelle Daten in IOCB0, da es diesen benutzt (normale Screen-Bearbeitung). Außerdem darf sie bei Betriebssystemänderungen nur zusammen mit DSKRDERR verschoben werden (s. dort)!

$c667 50791 $f39d 62365 GETBLOCK

Beim Booten von Cassette oder Diskette wird diese Routine für jeden Block einmal aufgerufen. Wird von Cassette gelesen, so hat CASSTART einen Wert ungleich Null und es wird über JUMPTAB+$2a die Prozedur CASREADBLK aufgerufen.

Handelt es sich um ein Disketten-Lesen, so wird DSKCMD mit dem Wert READCMD ($52) geladen, DSKUNIT auf 1 gesetzt und über JUMPTAB+$03 die SIO-Routine DISKINTERF aufgerufen.

$c67c 50812 $f3b2 CASBOOT

Wenn WARMFLAG auf Warmstart steht und Bit 1 von DOSAKTIV Eins ist, wird über CASINITC die Routine CASINIT aufgerufen; anderenfalls wird im Falle eines Warmstarts zum aufrufenden Programm zurückgekehrt.

Handelt es sich dagegen um einen Kaltstart, kann gebootet werden, wenn STARTTST mit einem Wert ungleich Null signalisiert, daß beim Eintreffen des Kaltstarts (also zum Beispiel beim Einschalten des Gerätes) die START-Taste gedrückt war.

Ist oder war sie es nicht, kehrt CASBOOT zum Aufrufer zurück.

Ansonst wird ein OPEN auf die Cassette ausgeführt, wobei der Timeout auf 'lang' gestellt wird. Dazu existiert das Register GAPTYPE. Wird GAPTYPE auf einen Wert zwischen Null und 127 gesetzt, wird das Lesen von Cassette nach recht kurzer Zeit unterbrochen, wenn kein neuer Block zu lesen ist. Bei einem größeren Wert (als signed-Wert: negativ!) wird wesentlich länger gewartet. Dies hat den Sinn, den Anfang eines Files auf Cassette abzuwarten.

Der eigentliche OPEN wird mit auf 1 gesetztem CASSTART über JUMPTAB+$2d und CASOPENIN ausgeführt, wobei danach gleich BLOCK1 aufgerufen wird. Ist der Block richtig gelesen, so wird von BLOCK1 DOSAKTIV mit dem Wert 2 zurückkehren. Ist dies der Fall, wird als letztes der Inhalt von DOSINITV nach CASINITV geladen.

$c6ae 50862 $f3e1 62433 CASINITC

Es wird ein indirekter Sprung über CASINITV ausgeführt.

$c6b1 50865 $edea 60906 DISKINIT

DSKTIMOUT wird mit $a0 initialisiert und die Sektorenlänge der Diskette in DSKSECLEN auf $0080 (also 128 Byte/Sektor) gesetzt. Hier beginnt die unter BOOT beschriebene Unterscheidung zwischen Block und Sektor interessant zu werden!

$c6c1 50881 $edf0 60912 DISKINTERF

Diese Prozedur erledigt sämtlich Initialisierungsaufgaben zum Lesen oder Schreiben eines Sektors von Diskette, beziehungsweise deren Formatierung oder einfach nur zum Lesen des Status' der Diskettenstation. Dazu werden die Variablen DSKDEVICE, DSKSTATUS, beim Status-Kommando DSKBUFFER, DSKTIMOUT sowie DSKBYTCNT mit den entsprechenden Werten initialisert. Diese Werte richten sich nach der Art des Kommandos. Als solche stehen dem Anwender folgende zu Verfügung:

Name	Code in DSKCMD	Erklärung

GET SECTOR $52 ('R')
Liest Sektor Nummer DSKAUX1 mit der Länge DSKBYTCNT von Diskette Nummer DSKDEVICE ein und schreibt ihn ab DSKBUFFER in den Speicher.

PUT SECTOR $50 ('P')
Schreibt Sektor Nummer DSKAUX1 mit der Länge DSKBYTCNT auf Diskette Nummer DSKDEVICE mit dem Speicherinhalt ab DSKBUFFER voll. Im Gegensatz zum 400/800 kann der 600XL/800XL dieses Kommando auch über das hier beschriebene Diskinterface verwerten. Bei den alten Geräten mußte 'getrickst' werden, um das Kommando anwenden zu können.

WRITE SECTOR $57 ('W')
Dieses Kommando führt zuerst die gleichen Operationen durch wie PUT SECTOR, vollzieht jedoch nach dem Schreiben noch ein Verify des betreffenden Sektors.

STATUS REQUEST $53 ('S')
Hier wird eine formatierte Statusangabe von der Diskette erwartet. Dieses Kommando setzt sich den Wert von DSKBUFFER selbst auf DEVICSTAT, es muß (und kann) also kein Pufferbereich für die Statusmeldung übergeben werden.

Das Kommando liefert, abgesehen von der Message DISK_READ_OK, vier Byte in DEVICSTAT bis DEVICSTAT+3 zurück:

DEVICSTAT	Kommandostatus
Bit 0	ist gesetzt nach einem ungültigen Kommando.
Bit 1	ist gesetzt nach dem Lesen eines defekten Sektors
Bit 2	ist gesetzt nach einem erfolglosen Schreibversuch über PUT SECTOR.

Bit 3 gibt im gesetzten Zustand an, daß die Diskette schreibgeschützt ist.

Bit 4 schließlich ist gesetzt, wenn das Laufwerk prinzipiell eingeschaltet ist und sich nach dem Status-Kommando zurückgemeldet hat.

DEVICSTAT+1 Hardwarestatus
Dieses Byte ist eine von der Diskette gesandte Kopie des Statusbytes des FD1771 Floppy-Disc-Controllers, der die gesamte Schreib-Leseverarbeitung innerhalb der Diskettenstation übernimmt. Obwohl der 1771 als recht veralteter FDC einige wesentlich anders gestaltete Hardwareeigenschaften besitzt als die neuen Prozessoren, sind die Statusworte bei den jeweiligen Typen gleich geblieben, so daß es auch bei Verwendung eines anderen FDC dieser Intelligenzstufe keine Schwierigkeiten bei der Interpretation dieses Bytes aus Diskettenstationen anderer Hersteller geben dürfte. Nur bei reinen Selbstbauprojekten, die FDCs verwenden, die wesentlich mehr Parameter zum Arbeiten benötigen als die 1771 / 1797- Typen, kann es hier Schwierigkeiten geben.

DEVICSTAT+2 Timeout-Wert
Dieser Wert gibt an, nach wieviel Sekunden eine Timeout-Kondition erkannt werden soll (s. allgemeine Beschreibung des SIO).

DEVICSTAT+3 dieses Byte ist noch ungenutzt.

FORMAT DISK $21 ('!')
Dieses Kommando erwartet als Parameter nur den Kommandocode, die DSKUNIT sowie einen freien Speicherbereich von 128 Byte ab DSKBUFFER. Bei diesem Kommando wird der Timeout erheblich höher gesetzt, da das Formatieren einer Diskette bekanntlich länger als 7 Sekunden dauert.

Ist die Formatierung beendet, werden an den Benutzer Informationen über den Zustand seiner Diskette zurückgeliefert. In DSKBYTCNT steht die Anzahl der defekten, also nach dem Formatieren nicht einwandfrei zu lesenden Sektoren, und ab DSKBUFFER stehen die Nummern der fehlerhaften Sektoren, falls vorhanden. Als letzter Eintrag in diesem Puffer steht $ffff.

Diese Eigenschaft des Atari, halbdefekte Disketten trotzdem weiter zu benutzen, also nur die defekten Sektoren kenntlich zu machen und den Rest zu bearbeiten, ist bei der Verwendung von Disketten durch den starken Preisverfall in den letzten Jahren völlig aus der Mode gekommen. Das soll keine negative Kritik an dieser Methode sein; eher im Gegenteil, denn die Tatsache, daß Atari hier nicht an Progammidee gespart hat, ehrt sie und erleichtert außerdem den softwaremäßigen Anschluß einer Harddisk. Wenn bei einem solchen Medium einmal ein Sektor (von bei einer 10MegaByte-Platte ca 82.000) defekt ist, wird nicht gleich die gesamte Platte weggeworfen, sondern nur dieser eine Sektor gekennzeichnet.

Die von DISKINTERF zu initialisierenden Werte für DSKTIMOUT und DSKBYTCNT werden der Routine jeweils in DSKTIMCON beziehungsweise DSKSECLEN übergeben. Die Pufferadresse DSKBUFFER, das Kommando DSKCMD sowie die Sektornummer (für Schreiben und Lesen) in DSKAUX1 (Low) und DSKAUX2 (High) müssen vor dem Aufruf von DISKINTERF in die Register eingetragen werden.

Als allgemeiner Rückgabewert ist im Y-Register ein Statuswort enthalten, das im Fehlerfalle negativ ist. Das N-Flag der CPU ist bei der Rückkehr aus der Routine entsprechend dem Ergebnis gesetzt.

$c748 51016 $ee6d 61037 PUTADRESS

DSKBUFFER wird nach BUFFERADR kopiert.

$c753 51027 $xxxx xxxxx LOADER

Diese Routine wird in Verbindung mit weiteren Lade- und Verwaltungsroutinen nicht für den momentanen Gerätepark der Firma Atari verwendet, sondern bietet dem Anwender die Möglichkeit, sich selbst Erweiterungen zu der Handlertabelle HATABS zu fertigen und diese neuen Geräte ebenfalls über IOCBs zu betreiben. Bekanntermaßen sind die zeichenorientierten Devices Editor, Screen, Keyboard, Lineprinter sowie Cassette am leichtesten über HATABS anzusprechen, es ist jedoch kein weiterer Platz in dieser Tabelle vorhanden für eventuelle weitere Geräte. Dieser Makel der alten 400/800-Serie wurde hier mit Hilfe von Routinen wie zum Beispiel LINK, UNLINK und anderen abgeholfen. Der Leser möge entschuldigen, wenn wir aufgrund fehlender, zu diesen Ports gehörender Hardware keinerlei Schwächen oder eventuelle Laufzeitfehler dieser Routinen hier feststellen können.

Zu Beginn dieser Routine werden RUNADR, HIUSED sowie ZHIUSED gelöscht.

Danch werden über GETBYTEA zwei Byte gelesen, wobei das erste Byte in HIBYTELD und das zweite Byte in SECLEN abgelegt wird. Generell gilt für diese Routine, daß sie mit gesetztem Carry und einem Y-Wert von $9c zurückkehrt, wenn GETBYTEA über das Carryflag einen Fehler meldet.

Wenn HIBYTELD ungleich $0b ist, wird die Routine aufge-

rufen, deren Adresse bei NEUVEKTOR + 2*HIBYTELD steht. Ist nach dieser Routine RECLEN gleich LCOUNT, so werden wieder zwei Byte nach HIBYTELD und SECLEN geladen und das Spiel geht von vorne los. Ansonsten wird nur noch ein Zeichen über GETBYTEA gelesen und indirekt über RUNADR die dort angegebene Routine aufgerufen, danach LCOUNT incrementiert und, wenn ungleich Null, zurückgesprungen zu der Stelle, an der RECLEN mit LCOUNT verglichen wurde. Ist LCOUNT hier jedoch Null, so terminiert die Routine.

War HIBYTELD gleich $0b, so wird GETBYTEA zweimal aufgerufen und die beiden gelesenen Zeichen als RUNADR beziehungsweise RUNADR+1 - Inhalte abgelegt, also die komplette Runadresse definiert. Ist RECLEN an dieser Stelle gleich Null, so wird die RUNADR gleich wieder gelöscht, ist RECLEN = $01, bleibt die RUNADR ungeändert und in den übrigen Fällen wird zu dem Wert von RUNADR der von LODADRESS addiert. In jedem Fall kehrt jedoch die Routine mit gelöschtem Carry-Flag und einem Y-Wert =$01 zurück zu dem aufrufenden Programm.

$c7dd   51165 $xxxx xxxxx   GETBYTEAC

Es wird ein indirekter Sprung über (LODGBYTEA) ausgeführt.

$c7e0   51168 $xxxx xxxxx   RUNADRC

Es wird ein indirekter Sprung über (RUNADR) ausgeführt.

$c7e3   51171 $xxxx xxxxx   PUTCHAR

Dieses Unterprogramm benötigt zwei Parameter: ein Informationsbyte im Accu sowie eines in LCOUNT.

Hat LCOUNT den Wert 0, so wird das Byte im Accu in

NEWLDTMP1 und NEWADRLOD abgelegt, bei einem LCOUNT-Wert von 1 in NEWLDTMP1+1 sowie NEWADRLOD+1. Im letzteren Fall werden noch einige Berechnungen ausgeführt, die sich im einzelnen nach dem Wert von HIBYTELD richten:

HIBYTEL	Ausgeführte 16-Bit-Berechnung
$00 oder $0a	NEWADRLOD := NEWLDTMP1 + LODADRESS
sonst	NEWADRLOD := NEWLDTMP1 + LODZLOADA

Danach wird ein 16bit großes Hilfsregister auf dem Stack eingerichtet, das den Wert

HIUSEDLOD + RECLEN - 2

enthält. Die danach ablaufenden Vergleiche richten sich wieder nach dem Wert von HIBYTELD:

HIBYTELD	16bit-Vergleich	Oper. wenn erfolgreich
$00 oder $0a	HILF = HIUSEDLOD	HIUSEDLOD := HILF
sonst	HILF = LODZHIUSE	LODZHIUSE := HILF

Ist Hilf allgemein größer als MEMTOP, so kehrt die Routine mit einem Y-Wert von $9D und gesetztem Negativ-Flag in der CPU zu der vorletzten Routine in der Prozedurhierarchie, also nicht zu dem eigentlichen Aufrufer zurück.

Hat LCOUNT weder den Wert 0 noch 1, so wird einfach das im Accu übergebene Byte in einer durch den Pointer LOADERTMP spezifizierten Adresse abgelegt. LOADERTMP wird berechnet durch LOADERTMP := NEWADRLOD + LCOUNT-2.

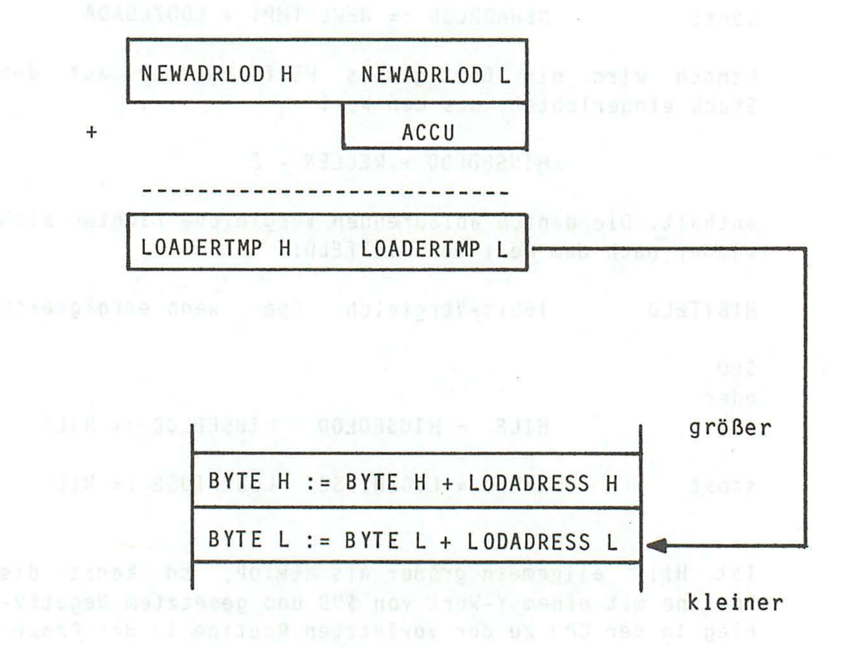
$c87b 51323 $xxxx xxxxx ADD28E

Es wird folgende Rechnung mit dem im Accu übergebenen Wert durchgeführt:

LOADERTMP := NEWADRLOD + Accu

(LOADERTMP) := (LOADERTMP) + LODADRESS *** 16bit-Oper.

Die nachstehende Grafik soll dies etwas veranschaulichen:

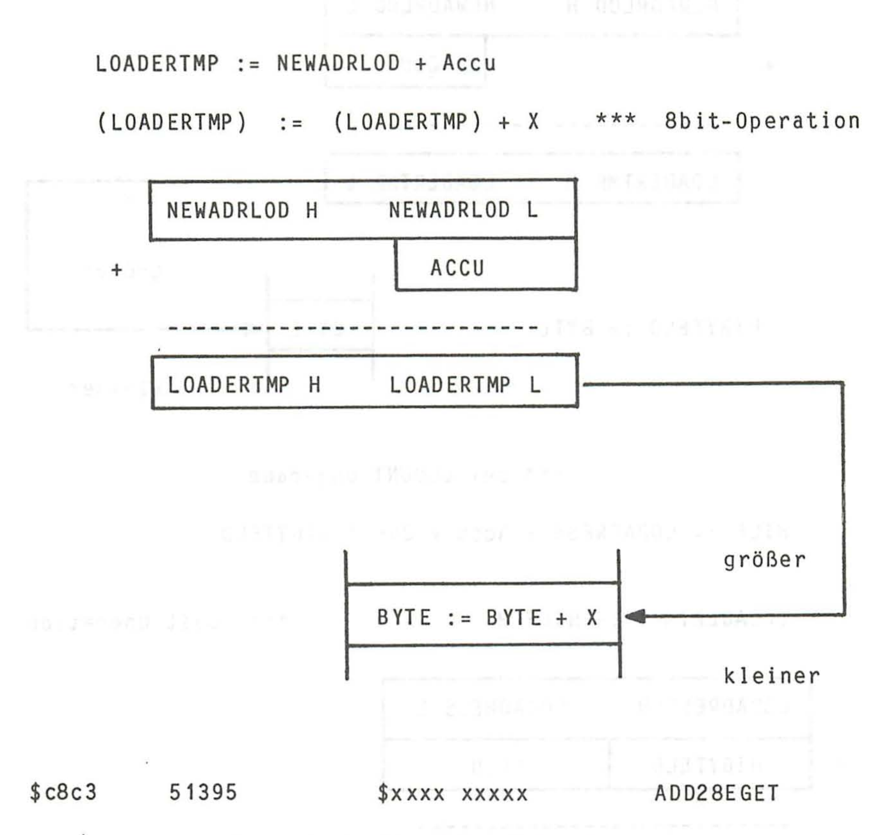


$c8a0 51360 $xxxx xxxxx ADD28EWRD

Hat HIBYTELD einen Wert größer oder gleich 4, so wird der Wert von X gleich dem Low-Byte von LOADADRESS, sonst gleich dem Low-Byte von LODZLOADA. Im Accu wird ein Byte übergeben, das folgendermaßen verwendet wird:

LOADERTMP := NEWADRLOD + Accu

(LOADERTMP) := (LOADERTMP) + X *** 8bit-Operation

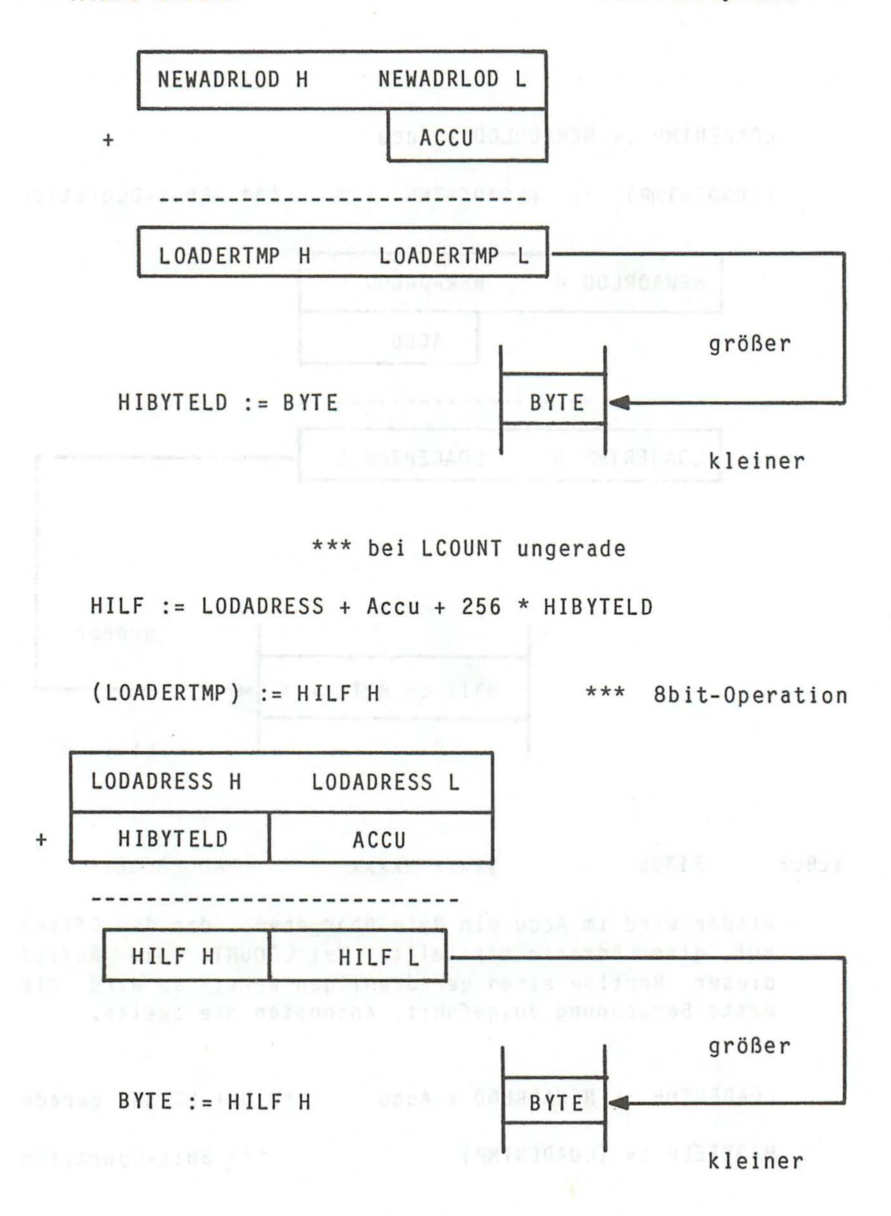


$c8c3 51395 $xxxx xxxxx ADD28EGET

Wieder wird im Accu ein Byte übergeben, das den Offset auf eine Adresse darstellt. Hat LCOUNT beim Aufruf dieser Routine einen geradzahligen Wert, so wird die erste Berechnung ausgeführt, ansonsten die zweite.

LOADERTMP := NEWADRLOD + Accu *** bei LCOUNT gerade

HIBYTELD := (LOADERTMP) *** 8bit-Operation


NEWADRLOD H
NEWADRLOD L
+
ACCU
LOADERTMP H
LOADERTMP L
größer
HIBYTELD := BYTE
BYTE
kleiner
*** bei LCOUNT ungerade
HILF := LODADRESS + Accu + 256 * HIBYTELD
(LOADERTMP) := HILF H
*** 8bit-Operation
LODADRESS H
LODADRESS L
+
HIBYTELD
ACCU
HILF H
HILF L
größer
BYTE := HILF H
BYTE
kleiner

$c8f2 51442 $xxxx xxxxx NEUVEKTOR

Diese Tabelle beinhaltet Ansprungpunkte für indirekte Aufrufe aus LOADER. Die Ansprungvektoren sind im Einzelnen:

PUTCHAR
PUTCHAR
ADD28EWRD
ADD28EWRD
ADD28EWRD
ADD28EWRD
ADD28E
ADD28E

$c90a 51466 $xxxx xxxxx SWITCHROM

Diese Routine schaltet das TestROM ein und startet es über JUMPTAB+$33 direkt in $5000 . Das Einschalten des ROMs geschieht über Rücksezten der PORT B7 - Leitung. Für die ordentliche Rückkehr ins Betriebssystem wird an dieser Stelle gleich COLDSTART auf den Wert $ff (also Kaltstart) gesetzt.

$c91a 51482 $xxxx xxxxx NEUINIT

*** ACHTUNG *** Diese Routine darf unter keinen Umständen während eines Programmablaufs aufgerufen werden; sie führt auf jeden Fall zum unkontrollierten Absturz der Maschine!

Sie ist vorgesehen für das Austesten später zu entwikkelnder Hardware und ruft unter bestimmten, im normalen Betrieb leicht zu erreichenden Bedingungen Routinen auf, die im Bereich des Mathe-ROMs abgelegt werden sollen. Da aber im Moment an dieser Stelle eben noch das Mathe-ROM liegt, springt die Routine hier 'in die Wüste'.

Trotzdem sei hier kurz die eigentlich gewünschte Funktion erläutert:

Es werden, beginnend mit bit 0, nacheinander sämtliche Bits in der Adresse NEUPORT gesetzt und nachgesehen, ob in $d803 der Wert $80 und in $d80b der Wert $91 steht. Ist dies der Fall, wird die Routine ab $d819 aufgerufen, wo später wahrscheinlich einmal für das Gerät, das sich eben gemeldet hat, ein IOCB und eine Handlereintragung eingerichtet werden sollen. Ist die Routine ab $d819 beendet oder ist eines der beiden Bytes nicht richtig gesetzt, wird die gleiche Abfrage mit dem nächsten Bit durchgeführt.

Zum Abschluß wird NEUPORT auf Null gesetzt.

$c941 51521 $e959 SIOINTERF

Diese Routine bildet den Einsprungpunkt für die SIO-Bearbeitung über Vektor JUMPTAB+$09.

Für uns ist im Moment eigentlich nur von Interesse, daß zuerst CRITICIO eingeschaltet wird, um von hier aus die Routine SIO aufzurufen und nach Beendigung der Bearbeitung das Y-Register mit dem Wert von DSKSTATUS zu laden, nachdem CRITICIO wieder gelöscht wurde.

Für den Fall, daß beim Eintritt in diese Routine das Register NEUVORHDN nicht Null ist, werden allerdings noch einige weitere Operationen durchgeführt. Zuerst wird das als Parameter an GETLOWEST zu übergebende X-Register mit $08 geladen und GETLOWEST aufgerufen. Kehrt die Routine mit gelöschtem Zero-Flag zurück, so wird X für die weitere Verwendung gerettet und $d805 aufgerufen (Mathe-ROM!). Ist das Carry-Flag als Status aus $d805 gelöscht, wird die eben beschriebene Prozedur mit decrementiertem X-Register wiederholt, ansonsten, oder wenn GETLOWEST mit Z=1 zurückkommt, wird SIO aufgerufen und weiter verfahren wie ohne diesen Exkurs.

$c97c 51580 $xxxx xxxxx NEUIOREQ

Dieser Programmteil wird angesprochen, falls von einem zukünftigen I/O-Device ein Interrupt kommen sollte. Für diesen Fall wird der Wert von NEUIODREQ auf dem Stack gerettet, um die Nummer der Position des niedrigsten gesetzten Bits des im Accu übergebenen Bytes nun in NEUIODREQ und gleichfalls in NEUPORT abzulegen. Danach wird kurzerhand die (noch) im Mathe-ROM liegende Routine $d808 aufgerufen, um nach deren Beendigung die alten Werte in NEUPORT und NEUIODREQ wiederherzustellen. Danach wird, wie beim I/O-Interrupt üblich, das gerettete X- und A- Register gepopt und mit RTI zum unterbrochenen Programm zurückgekehrt.

$c99f 51615 $xxxx xxxxx NEUDEVC1

Es wird Y mit dem Wert $01 geladen und CHECKNEWP aufgerufen.

$c9a4 51620 $xxxx xxxxx NEUDEVC3

Es wird Y mit dem Wert $03 geladen und CHECKNEWP aufgerufen.

$c9a9 51625 $xxxx xxxxx NEUDEVC5

Es wird Y mit dem Wert $05 geladen und CHECKNEWP aufgerufen.

$c9ae 51630 $xxxx xxxxx NEUDEVC7

Es wird Y mit dem Wert $07 geladen und CHECKNEWP aufgerufen.

$c9b3 51635 $xxxx xxxxx NEUDEVC9

Es wird Y mit dem Wert $09 geladen und CHECKNEWP aufgerufen.

$c9b8 51640 $xxxx xxxxx NEUDEVCB

Es wird Y mit dem Wert $0b geladen und CHECKNEWP aufgerufen.

$c9bd 51645 $xxxx xxxxx GETLOWEST

Diese Routine erwartet in X einen Wert zwischen 1 und 8. Je nach der Größe dieser Zahl wird mit dem 2**(8-X) ten Bit des Wertes von NEUVORHDN begonnen zu testen, ob es 1 ist. Mit anderen Worten wird mit dem niederwertigsten Bit begonnen, wenn X den Wert 8 hat und bricht die Routine ab, wenn X den Wert 0 hat. Wurde ein beliebiges Bit in NEUVORHDN gefunden, das den Wert 1 hat, so wird in NEUIODREQ und NEUPORT genau nur dieses Bit gesetzt. Beim Anschluß entsprechender Hardware an den 600XL/800XL hat diese Routine dann den pragmatischen Wert, für jedes, in NEUVORHDN gesetzte Bit einmal einen Request an das Peripheriegerät zu senden. Diese Routine wird von SIOINTERF aufgerufen, wenn NEUVORHDN ungleich Null ist, ansonsten wird sie von CHECKNEWP benutzt.

$c9d8 51672 $xxxx xxxxx NEUPORTERR

Auch diese Routine findet bei dem momentanen Hardwareangebot keine Verwendung. Sie erhält in Y einen Wert, der als Offset verwendet wird. Nachdem der Wert von $d80d+Y und der von $d80e+Y auf dem Stack abgelegt wurden, wird der Accu mit dem Wert von Adresse $024c und das X-Register mit dem von $024d geladen, um danach in Y den Immediatwert $92 zurückzugeben.

$c9ea 51690 $xxxx xxxxx CHECKNEWP

An diese Routine werden zwei Werte übergeben: einer im Accu und ein zweiter in X. Sie werden in $024c und $024d abgelegt und der momentane Zustand von CRITICIO gesichert, um dieses Flag danach auf den Wert 1 zu setzen.

Anschließend wird in einer Schleife GETLOWEST mit einem X-Wert von $08 aufgerufen. Kehrt GETLOWEST mit gelöschtem Zero-Flag zurück, so wird NEUPORTERR aufgerufen . Kehrt diese Routine wiedrum mit gesetztem Carry-Flag zurück, wird der Wert im Accu nach $024c geladen (was unsinnig erscheint, da NEUPORTERR diesen Wert gerade erst geladen hat!?!).

Kehrt GETLOWEST mit gesetztem Z-Flag zurück, das heißt ist kein weiterer I/O-Port vorhanden, so wird Y mit $82 geladen.

Unabhängig davon, welchen Wert bis hierhin GETLOWEST geliefert hat, werden NEUIODREQ und gleichzeitig NEUPORT auf Null gesetzt. Danach wird in CRITICIO der Anfangswert zurückgeladen, der Accu mit dem Wert aus $024c geladen, der Wert in Y nach $024d gelegt und entsprechend dieses Wertes die CPU-Flags gesetzt und zum Aufrufer zurückgekehrt.

Ist das Carry-Flag nach der Bearbeitung des Treibers Null, wird einfach die Schleife ein weiteres Mal durchlaufen, diesmal jedoch mit (von GETLOWEST) decrementierten X-Wert.

$ca2f 51759 $xxxx xxxxx BITMASK

Sie wird zum Ausblenden von Bits benötigt und enthält die Werte $80, $40, $20, $10, $08, $04, $02, $01 .

$ca37 51767 $xxxx xxxxx PREPLINK

Diese Prozedur ist zuständig für das Öffnen eines IOCB-Kanals, der für die neue, spätere Hardware zur Verfügung gestellt wird. Sie ruft INITLOAD, LINK+$06 und DEVS2PL3+$17 auf und springt dann mitten in die CIOMAIN-Routine, um das OPEN-Statement zu beenden. Tritt während dieser ganzen Bearbeitung ein Fehler auf, so wird ebenfalls mitten in CIOMAIN gesprungen, aber diesmal mit einem Y-Wert von $82, der CIOMAIN signalisieren soll, daß das angeforderte Device nicht existiert.

$ca65 51813 $xxxx xxxxx TABELLEN

$cb64 52068 $xxxx xxxxx CHECKFF

Diese Routine bildet die Checksumme (mit Carry!) über das Byte, auf das ZCHAIN zeigt und die 17 folgenden Byte. Hat die Checksumme den Wert $ff, so kehrt CHECKFF mit gesetztem Zero-Flag zurück, sonst ist das Flag gelöscht. Diese Routine testet also den Inhalt eines Handlereintrages, der über LINK beziehungsweise UNLINK verwaltet wird.

$cb73 52083 $xxxx xxxxx FREIO

Ab hier bis $cbff sind 141 freie Byte, die für Programmerweiterungen verwendet werden können. Aufpassen beim Systemstart mit der Checksummenbildung! Am besten läßt man einfach einmal die Checksummenroutinen durchlaufen und notiert die berechneten Werte für jeden der Blöcke (s. CHECKROM1 und CHECKROM2).

$cc00 52224 $xxxx xxxxx INTERCHAR

An dieser Stelle beginnt der zweite interne Zeichensatz

von Atari. Er wird eingeschaltet durch das Setzen von $CC in CHARBASE§ und ermöglicht dann die Verarbeitung von Sonderzeichen wie 'äöü#' und vielen anderen, anstelle der Grafiksymbole.

$d000 53248 $d000 53248 IO-Baugruppen

$e000 57344 $e000 57344 STANDCHAR

Dies ist der Standard-Zeichensatz, der durch das Setzen von $e0 in CHARBASE§ eingeschaltet wird. Er beinhaltet Grafiksymbole.

$e400 58368 $e400 58368 EDITORVKT

$e410 58384 $e410 58384 SCREENVKT

$e420 58400 $e420 58400 KBVKT

$e430 58416 $e430 58416 LPTVKT

$e440 58432 $e440 58432 HANDLERTB

Diese Adressen bilden eine Handlertabelle, die für jedes Device folgende Einträge enthält:

OPEN	- Routinenadresse
CLOSE	- Routinenadresse
GET	- Routinenadresse
PUT	- Routinenadresse
STATUS	- Routinenadresse
SPECIAL	- Routinenadresse
einen	Sprungbefehl zur jeweiligen POWERON - Initialisierung
ein	Hilfsbyte

Device	OPEN	CLOS	GET	PUT	STAT	SPEC	JMP	POWR	HILF
EDITOR	ef93	f22d	f249	f2af	f21d	f22c	4c	ef6e	00
SCREEN	ef8d	f22d	f17f	f1a3	f21d	f9ae	4c	ef6e	00
KB	f21d	f21d	f2fc	f22c	f21d	f22c	4c	ef6e	00
LPT	fec1	ff01	fec0	feca	fea2	fec0	4c	fe99	00
CASS	fce5	fdce	fd79	fdb3	fdcb	fce4	4c	fcdb	00

Die Tabelle enthält jeweils die entsprechende Ansprungadresse in hexadezimaler Form.

$e450 58368 $e450 58448 JUMPTAB

Diese Sprungtabelle ist die einzige Möglichkeit, Programme zu entwerfen, die sowohl auf dem 400/800 als auch auf dem 600XL/800XL laufen können. Sie beinhaltet alle wesentlichen Einsprungpunkte von Routinen, die nicht über indirekte Vektoren laufen können oder sollen. In diesem Buch finden Sie einige Referenzen auf eben diese Tabelle, wobei alle Ansprünge der Übersichtlichkeit halber auf JUMPTAB+$xx und nicht direkt auf die Sprungadresse bezogen sind.

Jeder Tabelleneintrag enthält genau einen Sprungbefehl.

$e450 58368 $e450 58368 JUMPTAB+$00

Sprung zu DISKINIT

$e453 58371 $e453 58371 JUMPTAB+$03

Sprung zu DISKINTERF

$e456	58374	$e456	58374	JUMPTAB+$06

Sprung zu CIOMAIN

$e459	58377	$e459	58377	JUMPTAB+$09

Sprung zu SIOINTERF

$e45c	58380	$e45c	58380	JUMPTAB+$0c

Sprung zu SETVBLVKT

$e45f	58383	$e45f	58383	JUMPTAB+$0f

Sprung zu SYSTEMVBL

$e462	58386	$e462	58386	JUMPTAB+$12

Sprung zu EXITVBL

$e465	58389	$e465	58389	JUMPTAB+$15

Sprung zu SIOINIT

$e468	58392	$e468	58392	JUMPTAB+$18

Sprung zu SENDENABL

$e46b	58395	$e46b	58395	JUMPTAB+$1b

Sprung zu NMIENABLE

$e46e 58398 $e46e 58398 JUMPTAB+$1e

Sprung zu CIOINIT

$e471 58401 $xxxx xxxxx JUMPTAB+$21

Sprung zu TESTROMEN

$e474 58404 $e474 58404 JUMPTAB+$24

Sprung zu RESETWARM

$e477 58407 $e477 58407 JUMPTAB+$27

Sprung zu RESETCOLD

$e47a 58410 $e47a 58410 JUMPTAB+$2a

Sprung zu CASREADBL

$e47d 58413 $e47d 58413 JUMPTAB+$2d

Sprung zu CASOPENIN

$e480 58416 $xxxx xxxxx JUMPTAB+$30

Sprung zu TESTROMEN

Dies ist kein Druckfehler, der Ansprung wird tatsächlich zweimal vorgenommen. Der erste Ansprung ist eigentlich ein Auffangen eines Fehlsprunges, da dort bei der 400/800-Serie ein Sprung nach Label SIGNON stand, bei dem die Message "ATARI COMPUTER - MEMO PAD" ausgedruckt worden ist und dann in eine Echo-Funktion gesprungen wurde. Diese Funktion wurde bei dem neuen

600XL/800XL durch das Memory-TestROM ersetzt, das nun von hier aus angesprungen wird.

Damit diese 'Auffüll'-Funktion theoretisch von der eigentlichen TestROM-Einschaltfunktion getrennt ist, wurden hier diese beiden, vielleicht nur im Moment gleichen Sprungvektoren eingesetzt.

$e483 58419 $xxxx xxxxx JUMPTAB+$33

Sprung zu TESTSTART

Dieser Ansprung ist natürlich nur dann sinnvoll, wenn auch das TestROM eingeschaltet, oder aber sein Inhalt in das RAM ab Adresse $5000 kopiert worden ist.

$e486 58422 $xxxx xxxxx JUMPTAB+$36

Sprung zu NEUDEVICE

$e489 58425 $xxxx xxxxx JUMPTAB+$39

Sprung zu UNLINK

$e48c 58428 $xxxx xxxxx JUMPTAB+$3c

Sprung zu LINK

$e48f 58431 $xxxx xxxxx CALLTAB

Hier liegen 6 Adressen in einer Tabelle, die auf NEUDEVC1 bis NEUDEVCB zeigen, respektive auf das Byte direkt vor diesen Adressen. Dies hat den Zweck, mit Hilfe dieser Tabellenwerte die entsprechenden Device-Routinen mit RTS aufrufen zu können.

siehe CIOJUMP 59124

Es dürfte allgemein bekannt sein, daß die 6502-CPU beim Rücksprung aus einer Prozedur auf dem Stack einen 16bit-Wert als Adresse verlangt, der auf das letzte abgearbeitete Byte der aufrufenden Routine zeigt, um dann die danach stehenden Befehle abzuarbeiten. Diesen Effekt kann man nun ausnutzen, um über eine Tabelle indirekt Routinen aufrufen zu können. Dazu muß einfach die aufzurufende Adresse-1 auf dem Stack abgelegt und ein RTS (ReTurn from Subroutine) ausgeführt werden.

$e49b 58523 $xxxx xxxxx NEUINITC

Es folgt ein direkter Sprung zu NEUINIT.

$e49e 58526 $xxxx xxxxx FREI1

Ab hier sind 35 Byte Programmspeicher noch nicht belegt. Beim benutzerseitigen Belegen des Platzes ist darauf zu achten, daß die ROM-Checksummen (s. CHECKROM1 und CHECKROM2) korrigiert werden!

$e4c0 58560 $xxxx xxxxx RTS

Hier steht ein völlig vereinsamtes RTS. Es wird von verschiedenen Routinen benutzt.

$e4c1 58561 $e4a6 58543 CIOINIT

Diese Routine löscht alle IOCBs dadurch, daß sie die Devicenamen CHANNELID ($0340, $0350 ...) auf $ff legt und die PUTBYTE-Zeiger ($0346+7, $356+7, ...) auf CIONOTOPN als Fehlermeldung legt.

$e4dc 58588 $e4c1 58561 CIONOTOPN

Wird versucht, Ausgaben über ein nichtexistentes Device

zu senden, so wird diese Routine angesprungen. Sie liefert in Y einen Wert $85 zurück, der der aufrufenden Schicht gleichzeitig mit dem gesetzten Negativ-Flag des CPU-Statusbytes einen DEVICE_NOT_OPEN-Error signalisiert.

$e4ef 58591 $e4c4 58565 CIOMAIN

Diese Hauptausgaberoutine erwartet zwei Eingabeparameter:

Erstens ein zu sendendes Character im Accu und zweitens die IOCB-Nummer * 16 des Kanals, an den das Zeichen geschickt werden soll. Der Multiplikator 16 kommt dadurch zustande, daß jedes IOCB 16 Byte belegt.

Zuerst werden das übergebene Zeichen und die IOCB-Nummer in IOCBCHARZ respektive in IOCBNUMZ abgelegt, um danach die IOCB-Nummer zu überprüfen, ob sie erstens nicht zu groß ist (da nur 8 IOCBs zur Verfügung stehen ist $70 die größte erlaubte Nummer), und zweitens, ob sie überhaupt durch 16 teilbar ist. Ist eine der beiden Bedingungen nicht erfüllt, so wird in Y der BAD_IOCB-Error durch den Wert $86 signalisiert.

Generell kann zu den CIO-Routinen gesagt werden, daß sie mit gesetztem Negativ-Flag zurückkehren, wenn ein Fehler eingetreten ist. Der Fehlercode kann dann am Wert von Y erkannt werden.

Danach wird der ab IOCBx liegende Block nach IOCBCHIDZ kopiert, also an die ausführbare Zero-Page-Stelle.

Hier werden nun wieder einige Abfragen bezüglich noch nicht verwendeter Konstrukte getätigt. Ist als Channel-Id IOCBCHIDZ der Wert $7f angegeben und IOCBCMDZ nicht $0c (POWER_ON_INIT) und HNDLRLOAD Null, so wird ebenso ein INVALID_CODE-Error signalisiert, als ob IOCBCMDZ kleiner als 3 (OPEN) wäre. Ist HNDLERLOAD nicht Null, wird PREPLINK aufgerufen und ein eventuell gemeldeter

Fehler nach 'oben' durchgereicht. Tritt kein Fehler auf, wird mit der Kommandobearbeitung fortgefahren.

Ist schon zu Beginn IOCBCMDZ gleich $0c, wird sofort zum CLOSE - Kommando gesprungen.

Sonst wird von hier aus je nach Kommando entweder CLOSE, CIOSTATSP, CIOREAD oder CIOWRITE aufgerufen.

Hat HNDLRLOAD einen Wert ungleich Null, wird die Routine SPECHANDL aufgerufen, sonst DEVS2PL3. Kehrt diese Routine mit Carry = 1 zurück, wird trotzdem noch einmal HNDLRLOAD angesprungen, sonst wird DEVICSTAT und DEVICSTAT+1 gelöscht und COMENT aufgerufen.

$e57c 58748 $e533 58675 CIOCLOSE

Hier wird, je nach an CIO übergebenem Parameter, der entsrechende IOCB auf FREI gesetzt und der dazugehörige CLOSE-Treiber aus der Tabelle berechnet und angesprungen.

$e597 58775 $e54e 58702 CIOSTATSP

Hier wird, ebenso wie in CIOCLOSE, der entsprechende Treiber für STATUS respektive SPECIAL-Kommando aufgerufen, wenn das angesprochene Device existiert. Wenn nicht, wird es nach Möglichkeit eingerichtet und dann der dazugehörige Handlerteil angesprungen.

$e5b2 58802 $e569 58729 CIOREAD

Zu Beginn wird getestet, ob dieser IOCB eventuell lesegeschützt ist. Dies ist der Fall, wenn in IOCBAUX1Z das Bit 2 nicht gesetzt ist. In diesem Fall würde CIOREAD mit einem WRITE_ONLY-Error zurückkehren. Lesegeschützte Geräte sind zum Beispiel Drucker.

Handelt es sich um ein lesbares Gerät, wird die noch zur Verfügung stehende Pufferlänge in IOCBBUFLZ (16 Bit) auf Null verglichen. Ist es Null, wird nur ein Character über den entsprechenden Handler eingelesen und zum Aufrufer zurückgekehrt.

Ist die Länge des Puffers dagegen ungleich Null, gibt es drei Möglichkeiten, das Lesen abzubrechen:

Die erste Abbruchbedingung sieht die Leseroutine, wenn sie einen Lesefehler bemerkt, also der aufgerufene Handler sein Carry-Flag setzt, bevor er zu CIOREAD zrückkehrt.

Als zweites kann der IOCB ein blockorientiertes Gerät sein (zum Beispiel Floppy). Dann wird das Lesen beendet, wenn die Pufferlänge nach dem Lesen des letzten Zeichens Null wurde.

Die dritte Möglichkeit ergibt sich bei zeilenorientierten Geräten, wie Druckers sie darstellen. Dabei wird so lange gelesen, bis ein NEWLINE ($9b) gefunden wird. Hat bis dahin der Puffer gereicht, ist alles in Ordnung, ansonsten sind alle Zeichen, die nach Füllen des Puffers kamen, ignoriert worden, und das letzte Zeichen des Puffers wurde mit dem NEWLINE überschrieben. Hat man zum Beispiel einen Puffer mit 5 Byte Länge und eine Zeile gelesen, die eine Länge von 8 Zeichen hatte, so bekommt man im Puffer übergeben:

1.CHR 2.CHR 3.CHR 4.CHR NL

Außerdem wird in diesem Fall im Y-Register der TRUNCATET_CODE übergeben.

Allgemein kann gesagt werden, daß jeder Block, gleich welchen Formats, nach dem vom Handler aus richtigen Lesen mit einem NEWLINE-Character endet. Dies war beim 400/800 nicht immer sichergestellt.

Die Entscheidung, ob es sich um ein block- oder zeichenorientiertes Gerät handelt, wird von Bit 1 von IOCBCMDZ gefällt. Ist es Null, handelt es sich um ein zeilenorientiertes Device.

Als letzte Operation wird in IOCBBUFLZ die korrekte, endgültige Pufferlänge übergeben.

$e61e 58910 $e5c9 58825 CIOWRITE

Auch beim Schreiben über einen IOCB wird zuerst geprüft, ob der Zugriff überhaupt gestattet ist. Dazu muß in IOCBAUX1 das 4. Bit gesetzt sein.

Ist das Schreiben erlaubt, so wird kontrolliert, ob die Pufferlänge Null ist. Ist dies der Fall, so wird lediglich ein einziges Zeichen übertragen. Dies ist kein Progammierfehler, sondern eine äußerst sinnvolle Einrichtung, über die schon der alte 400/800 verfügte: Da der Pufferlängenzähler von CIOxxxx verändert wird, ist es sinnvoll, davon auszugehen, daß beim Aufruf dieser Routine wenigstens ein Zeichen übertragen werden soll, und nicht wegen dieses einen Zeichens von der darüberliegenden Schicht der Pufferzähler gesetzt zu werden braucht. Die eigentliche Übertragung läuft über den PUT-Handlerteil des Gerätes.

Ist die Pufferlänge größer als Null, so wird das nächste Zeichen aus dem Puffer gelesen und übertragen, um danach über INCBFP und DECBUFL den Pufferpointer zu inbeziehungsweise den Pufferlängenzähler zu decrementieren. Ist beim Schreiben des Zeichens ein Fehler aufgetreten, wird dies entsprechend über das Y-Register mitgeteilt.

Handelt es sich bei dem Gerät um ein zeilenorientiertes Gerät, so wird so lange in CIOWRITE verharrt, bis das letzte übertragene Zeichen ein NEWLINE ($9b) war, ansonsten so lange, bis der Puffer leer ist. Ist bei einem zeilenorientierten Gerät das letzte im Puffer

befindliche Zeichen nicht NEWLINE, so wird dieses automatisch 'hinterhergeschoben'.

$e670 58992 $e4c4 58564 CIORETURN

Diese Abschlußroutine für alle CIO-Kommandos hat zwei Einsprungpunkte: CIORETURN und CIORETURN+$02 .

CIORETURN hat als einzige zusätzliche Aufgabe das Ablegen des an die Routine übergebenen Y-Wertes (Status der Operation) in IOCBSTATZ zu tun gegenüber CIORETURN+$02.

Dort wird dann die von Benutzer programmierte Pufferadresse wieder in IOCBBUFAZ eingetragen, um dann den kompletten Zero-Page-IOCB wieder in den Userbereich zu übertragen, damit der nächste Programmteil den IOCB benutzen kann. Man darf nicht vergessen, daß die CIO-Routinen nicht in den von Benutzer zu programmierenden Bereichen $0340 ... $03bf arbeiten, sondern sich den gerade aktiven Bereich von oben herunterkopieren, damit sie wesentlich schneller und effektiver in der Zero-Page arbeiten können (vergleiche hierzu entsprechende Literatur über Programmiermethoden bei der 6502-CPU). Nach dem Kopieren wird HNDLRLOAD auf Null gesetzt, um dann im Accu das letzte gelesene oder geschriebene Zeichen, in X der IOCB-Index und in Y der Status der letzten IOCB-Operation zurückgegeben werden. Durch das letzte Laden des Y-Registers ist das Negativ-Flag des Prozessor-Status-Wortes signifikant.

$e695 59029 $e63d 58941 COMPENTRY

Der Kanaloffset IOCBCHIDZ wird geprüft. Ist er größer als 33, so gilt die ID als falsch und COMPENTRY kehrt mit einem DEVICE_NOT_OPEN-Error zurück.

Ansonsten liegt bei HATABS+Y der Anfang des 3-Byte-Eintrages in HATABS. Das erste Byte enthält den Namen des Gerätes, das zweite und dritte einen Zeiger auf den

Beginn des Handlerkonstruktes, wie es zum Beispiel in HANDLERTB für die initiell schon vorhandenen Geräte gezeigt ist. Dieses Handlerkonstrukt beinhaltet Adressen zum Betrieb der CIO-Routinen.

Diese Anfangsadresse wird nun in IOCBAUX2 und -3 abgelegt. Danach wird der Offset des Handlers innerhalb des Handlerkonstruktes festgestellt, der sich aus dem Kommando und CMDTAB ergibt. Sodann wird der Pointer IOCBAUX2 und IOCBAUX3 so umgelegt, daß er direkt auf den Anfang der Handlerroutine zeigt.

Diese ganzen Vorgänge lassen sich am einfachsten an einem Bild erklären:

	HATABS-Eintrag	
	leer ' '	
	leer ' '	
	leer ' '	
	leer ' '	
	leer ' '	
	leer ' '	
	Keyboardtab 'K'	
	Screentab 'S'	
	Editortab 'E'	
	Castab 'C'	
Y-Register hat zum Beispiel den Wert $06	Printertab 'P'	HATABS

Also wird erst in IOCBAUX2 -3 ein Pointer auf den Eintrag 'Editortab' abgelegt und danach Editortab selbst.

$e6bb 59067 $e633 58931 DECBUFL

Hier erfolgt ein 16bit- Decrement auf den Pufferlängenwert in IOCBBUFLZ . Das Zero-Flag wird gesetzt, wenn nach dem Decrement der Zähler Null ist.

$e6c8 59080 $xxxx xxxxx DECBFP

Der Pufferzeiger in IOCBBUFAZ wird decrementiert. Es kann keine Aussage über Flags getroffen werden, dies ist jedoch genau wie bei INCBFP auch nicht sinnvoll, da alle Programmentscheidungen nicht über den Pufferzeiger, sondern ausschließlich den Pufferlängenzähler erfolgen.

$e6d1 59089 $e670 58992 INCBFP

IOCBBUFAZ wird 16bitweise um Eins erhöht.

$e6d8 59096 $e677 58999 SUBBFL

Hier wird die effektive Pufferlänge berechnet. Bei zeilenorientierten Geräten ist bekanntermaßen die zurückgelieferte Datenmenge unbekannt. Sie kann zum einen dadurch bestimmt werden, daß innerhalb des Puffers nach einem NEWLINE-Character ($9b) gesucht wird, andererseits stellt diese Routine hier die Anzahl der übergebenen Zeichen in IOCBBUFLZ als Rückgabewert zur Verfügung. Sie berechnet dazu einfach IOCBBUFLZ := IOCBBUFLx - IOCBBUFLZ , wobei x die Nummer des über IOCBCHIDZ spezifizierten Gerätes ist.

$e6ea 59114 $e689 59017 GOHANDLER

Diese Routine ruft über CIOJUMP zu dem vorher berechneten Devicehandler. Dazu wird erst das Y-Register mit dem Wert FUNCTION_NOT_IMPLEMENTED ($92) geladen, um

sicherzustellen, daß eine Fehlermeldung erfolgt, wenn der danach aufgerufene Handler vielleicht initiell nur auf ein RTS führt. Nach Rückkehr aus der Handlerroutine wird je nach überliefertem Y-Wert das Negativ-Flag gesetzt. Das Carry-Flag ist ebenfalls immer auf Eins gesetzt.

$e6f4   59124 $e693   59027   CIOJUMP

Hier wird genau das Verfahren angewendet, das in CALLTAB beschrieben ist. Es wird die Anfangsadresse-1 des Handlerunterprogramms auf dem Stack abgelegt und danach der Handler indirekt über RTS angesprungen. Die Tatsache, daß es sich bei den dafür benötigten Tabellenwerten um die Anfangsadressen-1 handelt, ist bei Erstellung neuer Handlervektorentabelle von äußerster Wichtigkeit. Leicht kann ein Handler dadurch funktionsuntüchtig werden, daß die falsche Adresse angegeben worden ist. Es ist auch zu bedenken, daß die ebenfalls bei jedem Deviceeintrag enthaltene Sprungadresse zu der Power-up Initialisierung direkt auf die entsprechende Routine zeigen muß, da der Power-up nicht indirekt über diese CIOJUMP-Funktion, sondern über den direkt vor der Adresse stehenden JMP-Opcode aufgerufen wird. Das Konzept ist nicht unbedingt leicht verständlich, aber nach einiger Gewöhnungszeit lassen sich auch dort Fehler leicht erkennen.

$e6ff   59135 $e6b8   59064   DEVS2PL3

Dieses Unterprogramm bestimmt den Wert von IOCBDSKNZ, also die aktuell zu benutzende Diskettenstationsnummer. Dazu wird die Nummer gelesen, die in dem Folgebyte auf (IOCBBUFAZ) steht. Liegt sie zwischen $31 und $39 (ASCII-Wert für die Ziffersymbole '1' .. '9'), so wird der zugehörige Wert $00 .. $09 in IOCBDSKNZ eingetragen, ansonsten ist der Defaultwert Eins.

$e712 59154 $e69e 59038 DEVICSRCH

Es wird das bei (IOCBBUFAZ) liegende Zeichen als Gerätename interpretiert und in der Tabelle ab HATABS gesucht. Wird ein übereinstimmender Eintrag gefunden, so wird der Offset des Eintrags mit dem richtigen Namen zu HATABS in IOCBCHIDZ notiert und das Carry-Flag gelöscht. Wird in keinem der HATABS-Einträge der gesuchte Name gefunden, signalisiert ein gesetztes Carry-Flag der aufrufenden Schicht einen Fehler.

$e72d 59181 $e6c9 59081 COMTAB

Diese Tabelle bildet eine Referenz für jedes IOCB-Kommando auf einen Vektor-Eintrag in der Handlerroutinentabelle ab HATABS.

Es führen die Kommandos		Auf den Vektor ab Byte
3	OPEN	0
4,5	GET_RECORD	4
6,7	GET_CHARACTER	4
8,9	PUT_RECORD	6
10,11	PUT_CHARACTER	6
12	CLOSE	2
13	STATUS_REQUEST	8
14	SPECIAL	10

$e739 59193 $xxxx xxxxx LINKSOMETH

Auch dieser Programmteil ist für spätere Verwendung weiterer Peripherieeinheiten vorgesehen, bei denen unter Umständen der in HATABS stehende Bereich zu klein

wird. Es ist nicht zu vergessen, daß bei HATABS noch Platz freigehalten wurde für den Betrieb von maximal 6 weiteren Geräten.

Alle diese LINK- und UNLINK-Routinen gehen davon aus, daß ab der Adresse STARTTST eine 19 Byte große Struktur steht. Interessant ist dabei, daß von einer Kassettenbenutzung offensichtlich bei Benutzung dieser Konstrukte ganz abgesehen werden soll. Dies ist auch ganz einsichtig, da die beiden an diesen Stellen stehenden Adressen nur zum Booten über Cassette benutzt werden. Geht man nun davon aus, daß dieser immense Softwareaufwand nur dafür getrieben wird, exclusive Hardware an den Atari anzuhängen, dürfte klar sein, daß die dafür vorgesehene Software nicht über Cassette gebootet wird. Als entsprechend exclusive Hardware war zum Beispiel in Mundpropaganda ein CP/M-Subsystem angekündigt worden; gesehen haben wir es bis heute leider noch nicht.

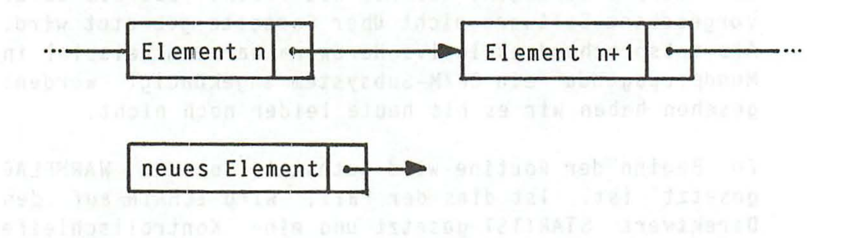
Zu Beginn der Routine wird getestet, ob das WARMFLAG gesetzt ist. Ist dies der Fall, wird ZCHAIN auf den Direktwert STARTTST gesetzt und eine Kontrollschleife begonnen.

Bevor jedoch diese Schleife und ihre Ausgangspunkte besprochen werden können, sollte ein wenig Theorie betrieben werden bezüglich der Verwendung linearer Listen beim Atari 600XL/800XL.

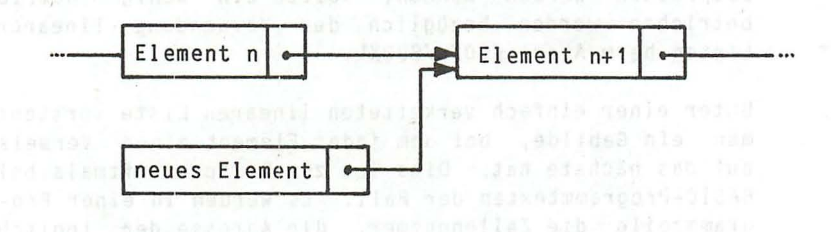
Unter einer einfach verketteten linearen Liste versteht man ein Gebilde, bei dem jedes Element einen Verweis auf das nächste hat. Dies ist zum Beispiel oftmals bei BASIC-Programmtexten der Fall. Es werden in einer Programmzeile die Zeilennummer, die Adresse der logisch hierauf folgenden Zeile und danach die eigentlichen Daten abgelegt. Hat man nun einen Zeiger auf die erste Zeile, findet man über diese die zweite, dann die dritte usw.. Das Verfahren ist vielleicht nicht unbedingt das schnellste, es hat jedoch in Punkto Verarbeitung einige wesentliche Vorteile. So muß zum Beispiel beim Einschieben einer neuen Zeile der gesamte, logisch

dahinter stehende Text nicht nach hinten verschoben werden, sondern es wird lediglich ein Zeiger so umgelegt, deß er auf das neue Element zeigt und der Zeiger des neuen Elements auf das Element, auf das der Vorgänger zuletzt zeigte. Damit liegt die neue Zeile logisch mitten im Text, wo sie jedoch physikalisch (also im Speicher) liegt, ist völlig egal.

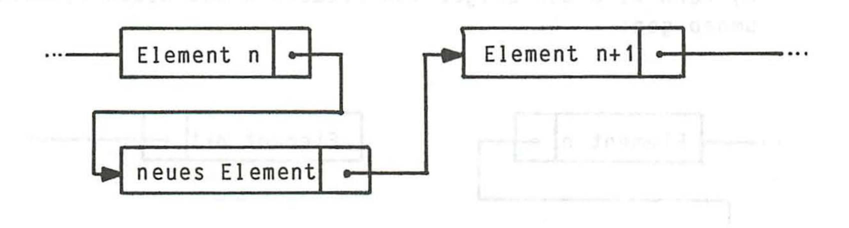
1) Es existiert eine lineare Liste, aus der ausschnittweise zwei Elemente gezeigt werden. Genau zwischen diese beiden Elemente soll ein weiteres eingefügt werden:



2) Dazu wird zuerst der Zeiger von Element n kopiert in den Zeiger für das neue Element:



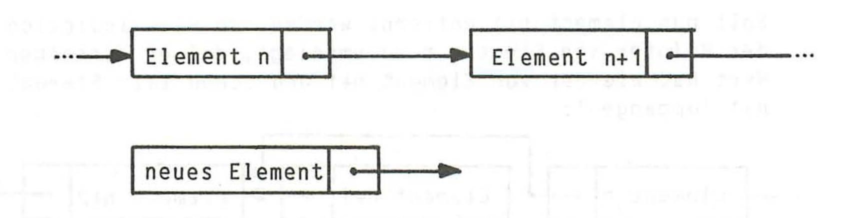
3) Nun wird der überflüssig gewordene Zeiger von Element n so umgelegt, daß er auf das neue Element zeigt. Von da ab ist das neue Element in der linearen Liste enthalten.



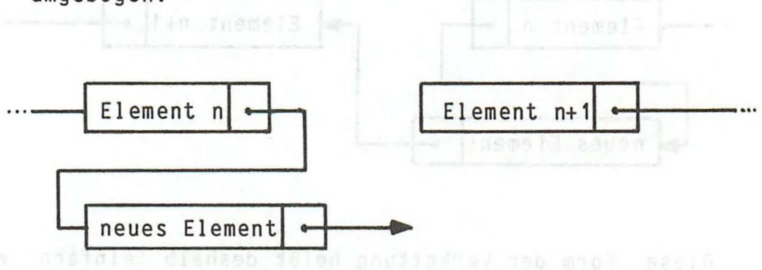
Diese Form der Verkettung heißt deshalb 'einfach verkettet', weil immer nur ein Zeiger (oder neudeutsch: Link, Verweis) auf den Nachfolger (häufig als successor bezeichnet) existiert, nicht jedoch ein Link auf den Vorgänger, der in der englischsprachigen Literatur Predecessor genannt wird. Solch eine Verkettung wäre dann unter dem Begriff 'doppelt verkettete Liste' in der Informatik bekannt.

Sicher dürfte klar sein, daß man nicht einfach ein solches Element aus der Liste löschen kann beziehungsweise, daß ein genaues Protokoll für das Einfügen (Linken) eines neuen Elements beziehungsweise das Entfernen (Unlinken) eines bestehenden Elementes einer verketteten Liste eingehalten werden muß. Würde zum Beispiel in unserem obigen Beispiel erst Schritt 3 und dann Schritt 2 gemacht, wären alle Elemente ab Element n+1 unrettbar verloren:

Die Ausgangsposition bleibt wie gehabt.



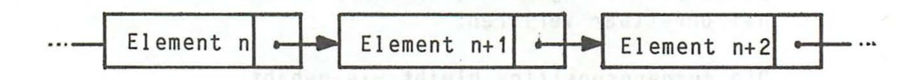
2) Dann wird der Zeiger von Element n zum neuen Element umgebogen:



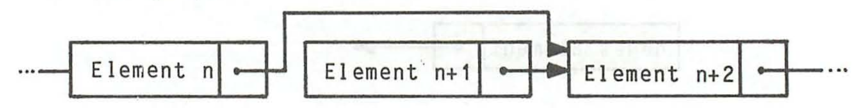
Schon haben wir den Effekt, daß nun der Zeiger des neuen Elements nicht auf eine sinnvolle Stelle zeigt und auch nicht mehr auf das Element n+1 umgebogen werden kann, denn wo hätte vorher notiert werden sollen, wo Element n+1 steht? Die Adresse stand explizit im Link des n. Elements und wurde schlichtweg 'vermurkst'.

Beim Unlink sind die Probleme etwas anderer Natur:

Nehmen wir an, wir hätten eine lineare Liste, von der wir drei irgendwo in der Mitte liegende Elemente betrachten:



Soll nun Element n+1 entfernt werden, so wird lediglich der Pointer von Element n so umgelegt, daß er denselben Wert hat wie der von Element n+1 und schon ist Element n+1 'umgangen':



Das Problem hierbei ist, sich zu merken, welche Speicherbereiche nun noch von aktuellen Listenelementen gefüllt sind und welche im Gegensatz dazu angefüllt sind mit nicht mehr referenzierten Elementen. Diese Berechnungen anzustellen ist relativ zeitaufwendig und wird deshalb beim Atari weitmöglichst umgangen. Bei anderen Systemen (zum Beispiel dem für Atari erhältlichen MicroSoft-Basic) ist die gesamte Stringverarbeitung auf dynamischer Speicherverwaltung aufgebaut. Der Ausdruck dynamisch besagt, daß nicht ein konstanter Speicherraum belegt wird, sondern immer nur gerade das, was benötigt wird. Bei Verwendung sehr vieler, sehr kleiner Strings passiert es dann ab und an, daß man glaubt, der Rechner 'stehe', obwohl er dann nach zwei bis drei Minuten weiterrechnet. Dann war er in der sogenannten Garbage-Collect-Routine, die den String-Müll aufsammeln sollte.

Ergänzend sollte noch erwähnt werden, was mit dem Pointer des letzten Elementes geschieht. Dieser zeigt nun nicht wirr 'in die Luft', sondern er bekommt einen definierten Wert zugewiesen, der allgemein NIL (Aussprache wie bei dem Fluß) genannt wird. Er ist wie bei Atari in dem meisten Fällen gleich Null, es gibt jedoch auch andere Computerkonzepte, die kaum noch etwas mit unseren Nullen und Einsen zu tun haben, so daß man ruhig die Bezeichnung NIL verwenden sollte.

Nach diesem kurzen Exkurs über dynamische Datenverwaltung zurück zu unserer gerade begonnenen Schleife. Bei diesen hier verwendeten Elementen sitzt der Link eines jeden Elements in Byte Nummer 18 und 19, bezogen auf die Startadresse des jeweiligen Elementes.

Zeigt zum Beispiel ZCHAIN auf ein beliebiges Element, so verstehen wir unter dem Konstrukt ZCHAIN.LINK eben den Link des Elementes, auf das ZCHAIN zeigt.

```
ZCHAIN          Byte 0
                Byte 1
                  .
                  .
                  .
                Byte 18     diese beiden Bytes sind in die-
                Byte 19     sem Fall ZCHAIN.LINK
```

Die Schleife terminiert (Ausdruck bei Schleifen, Funktionen und Prozeduren, wenn sie durch interne Ereignisse beendet werden) zum Beispiel, wenn ZCHAIN.LINK den Wert NIL liefert, es sich also bei dem behandelten Element um das letzte der Liste handelt.

Ansonsten wird das nächste Element geprüft. Dies geschieht einfach durch Zuweisung:

ZCHAIN := ZCHAIN.LINK

Dadurch zeigt nun ZCHAIN auf das Folgeelement. Bei diesem wird nun geprüft, ob die Quersumme alle Einträge vom 0. bis zum 17. Byte gleich $ff ist (CHECKFF). Ist sie es nicht, terminiert die Schleife ebenfalls.

Ansonsten wird das Carry-Flag gesetzt (über CALLVKTC1) und, ebenfalls darüber, CALLVKT angesprungen, was wiederum veranlaßt, daß ein Befehlsbyte angesprungen wird, das an Byte 12 der durch ZCHIAN bezeichneten Struktur steht. Bitte erinnern Sie sich, daß die Handlertabellen ab EDITORVKT ebenfalls ab Byte 12 einen Sprungbefehl auf die Initialisierungsroutine des entsprechenden Handlers hatte. Somit dürfte die Funktion und der Aufbau der hier beim 600XL/800XL neu eingeführten Strukturen klar sein: Sie sind funktional identisch mit den Handlertabellen ab EDITORVKT und vom Aufbau unterscheiden sie sich lediglich durch die noch weiter benötigten Parameter zur Listenverwaltung.

Kehrt dieser Initialisierungshandler nun mit gesetztem Carry-Flag zurück, so terminiert die Schleife eben-

falls, ansonsten wird der oben beschriebene NIL-Test bezüglich des Folgelinks nun auf das neue Element bezogen.

War zu Beginn der Routine WARMSTART gelöscht, wird CHAINLINK auf NIL gesetzt und mit der Initialisierung der IOCBs weiter unten im Programm begonnen.

$e76e 59246 $xxxx xxxxx

Nach Aufruf von DCBINIT, bei dem IOCB 0 mit Daten für ein Drive mit der Nummer 1 initialisiert wird, findet ein Aufruf von SIOINTERF über JUMPTAB+$09 statt.

Kehrt das SIO-interface mit negativem Status zurück, so gilt dies als Fehler ähnlich den CIO-Routinen und dieser Fehler wird sofort 'nach oben' durchgereicht.

War die Übertragung in Ordnung, wird CHAINTP1 mit der Summe aus MEMLO und dem Devicestatus DEVICSTAT geladen.

Ist nach diesen Operationen MEMTOP kleiner als CHAINTP1, werden DSKAUX1 und DSKAUX2 mit $4e geladen und DCBINIT aufgerufen. Danach wird zum Beginn der Routine zurückgesprungen.

Ist MEMTOP größer oder gleich CHAINTP1, wird der Wert von CHAINTMP gerettet, um danach CHAINTMP mit MEMLO zu laden. Hiernach wird INITLOAD mit dem geretteten Low-Byte aus CHAINTMP im Accu aufgerufen.

Kehrt diese Routine mit gesetztem Negativ-Flag zurück, wird LINK aufgerufen. Im anderen Fall wird, abhängig vom Carry-Flag, bei gelöschtem Carry komplett neu mit dieser Routine begonnen, sonst wird dort weitergemacht, wo DCBINIT mit $4e in DSKAUX1 und DSKAUX2 aufgerufen wird.

Der einzige Ausgangspunkt aus dieser Schleife ist also der negative Rückgabewert aus dem SIO-Interface.

$e7be 59326 $xxxx xxxxx DCBINIT

Es werden die Daten aus TABSIOINI in den Disk-Control-Block ab DSKUNIT geladen, der im Accu übergebene Wert nach DSKAUX1 und der in Y übergebene Wert in DSKAUX2 abgelegt, um danach SIOINTERF über die Sprungtabelle bei JUMPTAB+$09 anzuspringen.

$e7d4 59348 $xxxx xxxxx TABSIOINI

Hier stehen die Initialisierungswerte für den DCB (s. DCBINIT). Sie lauten im einzelnen:

DSKDEVICE	$4f	ist zu ignorieren
DSKUNIT	$01	Laufwerk 1
DSKCMD	$40	SIO-CMD GET_DATA
DSKSTAT	$40	ist zu ignorieren
DSKBUFFER	$02ea	ist DEVICSTAT, also normaler Ablagebereich für Status
DSKTIMOUT	$001e	entspricht 30 Sekunden Wartezeit bis Timeout
DSKBYTCNT	$0004	Die Statusinformation besteht aus vier Byte

$e7de 59358 $xxxx xxxxx INITLOAD

An diese Routine wird im Accu ein Wert für CHAINTP1H übergeben, der als erstes dort auch abgelegt wird. Der

Low-Teil CHAINTP1L wird auf Null und TEMP3 auf $ff gesetzt. Ist Bit 0 von CHAINTMP Eins, ist also der Vektor CHAINTMP ungerade, so wird CHAINTMP incrementiert. Damit ist sichergestellt, daß nach dieser Routine CHAINTMP auf eine gerade Adresse zeigt.

Dann wird LODADRESS mit dem Wert von CHAINTMP, LODGBYTEA mit der Adresse INCLOAD und LODZLOADA mit dem Wert $80 geladen, um danach LOADER aufzurufen.

$e816 59414 $xxxx xxxxx INCLOAD

Nach Incrementieren von TEMP3 wird, falls das Ergebnis nicht Null ist, der Accu mit dem Byte ($037d + Wert von TEMP3) geladen sowie das Carry-Flag gelöscht.

Ist TEMP3 nach dem Decrement Null, so wird es auf $80 gesetzt und DCBXINIT aufgerufen, was den DCB mit Werten für ein NOTE_SECTOR-Kommando lädt und das SIO-Interface aufruft. Ist der Note-Vorgang positiv verlaufen, wird ebenfalls der Accu mit dem Byte ($037d + Wert von TEMP3) geladen und das Carry gelöscht.

$e833 59443 $xxxx xxxxx DCBXINIT

Hier wird, analog zu DCBINIT, der Disk-Control-Block mit vorgefertigen Werten aus SIOTAB geladen und über JUMPTAB+$09 das SIOINTERF aufgerufen.

$e851 59473 $xxxx xxxxx SIOTAB

Hier stehen die Initialisierungswerte für den DCB für das NOTE-Kommando (s. DCBXINIT). Sie lauten im einzelnen:

DSKDEVICE	$00	ist zu ignorieren
DSKUNIT	$01	Laufwerk 1

DSKCMD	$26	NOTE_SECTOR
DSKSTAT	$40	ist zu ignorieren
DSKBUFFER	$03fd	Hier beginnt ein sonst nicht vom OS belegter RAM-Bereich.
DSKTIMOUT	$001e	entspricht 30 Sekunden Wartezeit bis Timeout
DSKBYTCNT	$0080	Es werden 128 Byte gelesen.

$e85d 59485 $xxxx xxxxx SEARCHLIS

In A und Y wird der High- beziehungsweise der Low-Teil einer Listenelementadresse übergeben, nach der hierbei ab Adresse STARTTST gesucht werden soll. Als Rückgabewerte sind ebenfalls der Accu, das X-Register und das Carry-Flag zu beachten. Diese drei Rückgabemechanismen haben folgende Bedeutung: das Carry-Flag gibt darüber Auskunft, ob der gesuchte Vektor gefunden wurde und ob A und X relevante Werte enthalten.

Damit gib es nur zwei Möglichkeiten:

1) Carry = 0 : Der gesuchte Vektor wurde gefunden und in A (High-Byte) und X (Low-Byte) zurückgegeben. ZCHAIN zeigt auf die Struktur, deren Link gleich dem gesuchten Pointer ist.

2) Carry = 1 : Entweder hatte ein Link den Wert NIL oder bei einer der untersuchten Strukturen fand sich eine, deren Checksumme nicht den Wert $ff hatte (Aufruf von CHECKFF!).

Im ersten Fall gibt sich zum Beispiel folgendes Bild, wenn nach LINK(n+1) gesucht wurde:

$03f9	Element1	Element2	Element n	Element n+1
	LINK(2)	LINK(3)	LINK(n+1)	

ZCHAIN

$e894 59540 $xxxx xxxxx CALLVKTC1

Es wird mit gesetztem Carry-Flag mitten in die LINK-Routine gesprungen, um als erstes über CALLVKT den Power-up-Handler anzuspringen.

$e898 59544 $xxxx xxxxx LINK

Diese Routine reiht ein neues Strukturelement, dessen Anfangsadresse im Accu (High-Wert) und Y (Low-Wert) übergeben wird, an das Ende der bestehenden Liste ein. Dazu wird als erstes das letzte Element der bestehenden Liste gesucht (Aufruf von SEARCHLIS mit A=Y=0) und dieser NIL-Wert so abgeändert, daß dieser Link nun auf das neue Element zeigt (s. LINKSOMETH!). Danach wird der Link des neuen Elements auf NIL gesetzt und direkt an das 12. Byte der neuen Struktur gesprungen, was zur Folge hat, daß ein dort stehender Sprungbefehl die Programmkontrolle an die Power-up-Routine des neu zu installierenden Devices übergibt. Kehrt diese Routine mit gesetztem Carry zurück, so wird dieses Element gleich wieder entfernt (Aufruf von UNLINK mit den alten Accu- beziehungsweise Y-Werten) und zum Aufrufer zurückgekehrt.

Ist dies nicht der Fall, wird geprüft, ob die aufrufende Routine ein gesetztes oder ein gelöschtes Carry-Bit übergeben hat. War es gelöscht, so werden das 16. und das 17. Byte der neuen Struktur gelöscht (MINTLK und

GINTLK). Danach wird, unabhängig des übergebenen Carry-Flags, MEMLO auf einen neuen Wert gesetzt:

MEMLO := MEMLO + ZCHAIN.MINTLK (16bit-Operation).

Somit kann von der aufrufenden Routine bestimmt werden, ob nach Einhängen des neuen Elements MEMLO verändert werden soll oder nicht. Ist das Carry gesetzt, wird MEMLO verändert, sonst nicht. Das hat speziell dann Sinn, wenn ganz zu Systembeginn diese Routinen aufgerufen werden und sich dann alle Programme wie DOS, BASIC und ähnliche, nach dem neuen, verschobenen MEMLO richten (MEMLO gibt die niedrigste, von dem Betriebssystem nicht mehr benötigte Adresse in RAM an).

Danach wird die Checksumme der neuen Struktur berechnet (CHECKFF) und das Ergebnis im Byte 15 der Struktur abgelegt. Das Byte 15 war auch schon zum Beispiel bei EDITORVKT ein Hilfsbyte - jetzt hat es eine Funktion übernommen.

Es ist an dieser Stelle erwähnenswert, daß CHECKFF neben der Z-Flag-Aussage noch im Accu die Differenz der Checksumme zum Wert $ff zurückgibt. Dieser Wert kann nun eingesetzt werden, damit bei weiteren Überprüfungen die Checksumme richtig ist.

$e900 59648 $xxxx xxxxx CALLVKT

Es wird an das Byte 12 der Struktur gesprungen, auf die ZCHAIN zeigt:

		Höhere Adressen
	19	
	18	
	17	
	16	
Checkfill	15	
High-Byte	14	

	Low-Byte 13	
	JMP ($4c) 12	
	11	
	10	
	09	
	08	
	07	
	06	
	Handler- 05	
	adressen 04	
	03	
	02	
	01	
ZCHAIN	00	Niedrigere Adressen

$e912 59666 $xxxx xxxxx SETBVBLVKC

Diese Adresse beinhaltet lediglich einen Sprung nach SETVBLVKT.

$e915 59669 $xxxx xxxxx UNLINK

Über SERACHLIS wird das in Accu und Y spezifizierte Listenelement in der linearen Liste ab STARTTST gesucht. Wird es nicht gefunden, kehrt die Routine mit gesetztem Carry-Flag zurück.

Das gefundene Element wird entweder gelöscht, wenn COLDSTART gesetzt, also ungleich Null ist, oder wenn erstens das Byte 16 und das Byte 17 Null sind und zweitens die Quersumme über die Struktur den Wert $ff liefert (CHECKFF). In eben diesen Fällen wird das Element auf die Art entfernt, die in LINKSOMETH eingehend beschrieben wird.

War das Element zu löschen, kehrt UNLINK mit gelöschtem, sonst mit gesetztem Carry-Flag zurück.

$e959 59737 $xxxx xxxxx JMPSIOINT

Es erfolgt ein direkter Sprung zum SIOINTERFace.

$e95c $59740 $e944 59716 SIOINIT

Diese Routine ist zuständig für die Initialisierung des POKEY. Es werden der Cassettenrecordermotor und der Ton abgeschaltet. Weiterhin wird die NOT_COMMAND-Leitung des seriellen Busses auf High (inaktiv) gelegt.

$e971 59761 $e959 59737 SIO

Diese Prozedur übernimmt sämtliche Koordinierungsaufgaben zur Steuerung des seriellen Busses.

Zuerst wird, um Fehlinterpretationen von Interrupts zu vermeiden, CRITICIO auf Eins gesetzt und unter DSKDEVICE nachgesehen, ob es sich bei dem angegebenen Gerät um die Cassette handelt. Ist dies der Fall, wird zu CASENTER, dem Cassettenbearbeitungsprogramm, gesprungen. Diese Unterscheidung ist an dieser Stelle wesentlich, da alle anderen Geräte außer der Cassette einen internen Kontroller besitzen, der ihnen 'Intelligenz' verschafft.

Haben wir es mit solch einem intelligenten Gerät zu tun, wird CASFLAG auf Null gesetzt, um nach Rückkehr aus der Lese- beziehungsweise Schreibroutine richtig reagieren zu können. Danach werden DRETRY und CRETRY mit $01 beziehungsweise $0d initialisiert und die Übertragungsgeschwindigkeit auf ca. 19.200 Bit/Sekunde festgelegt. Letztere Festlegung erfolgt über das Programmieren der Audiofrequenzkanäle 3 und 4, die mit $28 beziehungsweise $00 geladen werden.

Danach werden die Geräte-Identifikation, die sich aus der Geräteart und der Gerätenummer-1 zusammensetzt, das Kommando sowie das erste und das zweite Hilfsbyte

DSKAUX1 und DSKAUX2 in die entsprechenden Variablen ab CMDDEVIC geladen, um danach DSKBUFPTR auf CMDDEVIC zeigen zu lassen, damit die danach aufgerufene Kommandoroutine weiß, an welcher Stelle sie das zu bearbeitende Kommando findet. BUFENDPTR zeigt auf die Stelle direkt hinter CMDAUX2.

Sind diese Vorbereitungen erledigt, wird über PORT B die KOMMANDO auf Low gelegt, um allen Geräten zu sagen: "ACHTUNG, es kommt eine neue Geräteadresse!". Eben dieses Senden wird von der Routine SENDINIT übernommen. Hat das angesprochene Gerät einen defekten Status zurückgemeldet oder sich gar ein völlig anderes Gerät gemeldet, so wird ein Fehlercode in ERRFLAG zurückgegeben, der nicht Null ist. In diesem Fall, oder wenn in Y notiert wurde, daß das Gerät ein NEGTIV_ACKNOWELEDGE dadurch signalisierte, daß es Y auf Null setzte, wird CRETRY decrementiert und, falls nicht Null, erneut versucht, das Gerät anzusprechen. Gelingt dies nicht, findet eine Fehlermeldung an das aufrufende Modul statt.

Hat sich jedoch das Gerät ordentlich zurückgemeldet, wird DSKSTATUS ausgelesen und überprüft, ob es einen positiven Wert aufweist. Ist dies der Fall, so wird vor dem ersten Senden oder Lesen eines Datensatzes noch gewartet, bis das Device sein vorhergehendes Kommando völlig abgearbeitet hat, oder ein Timeout-Fehler auftritt. Danach wird ein Kommando-Datensatz an das Gerät gesendet. Der Disk-Control-Block muß die entsprechenden Parameter für das Senden beinhalten.

Nach dem ordentlichen Abarbeiten dieses Kommandos zeigt DSKSTATUS an, ob eventuell noch Daten von dem Gerät zu lesen sind. Ist dies der Fall, werden sie über RECEIVE eingelesen.

Treten während der gesamten Bearbeitung keine Fehler auf, so wird in DSKSTATUS eine $01 zurückgegeben, ansonsten der Fehlercode. Fehlercodes sind immer negativ und das N-Flag ist bei Rückkehr aus SIO gesetzt.

Auf SIO zeigt JUMPVKT+$09.

$ea37 59959 $ea1a 59930 WAIT

Dieses Modul wird nach dem Absetzen eines Kommandos an ein intelligentes Gerät benutzt, um auf die Rückmeldung zu warten.

Diese Rückmeldung kann zum Beispiel sein, daß das Gerät das Kommando nicht ausführen kann (zum Beispiel Schreiben auf schreibgeschützte Diskette) oder aber eine Positiv-Meldung, die hier durch die COMPLETE-Message dargestellt wird.

Kommt eine negative Rückmeldung oder kommt bis zum Timeout keine Rückmeldung, so wird das Y-Regster mit $00 geladen und in ERRORFLAG der entsprechende Fehler zum aufrufenden Modul zurückgemeldet.

$ea88 60040 $ea6b 60011 SEND

Nach Aufruf von SENDENABL wird die allgemein interruptgesteuerte Ausgabe dadurch angestoßen, daß das erste Byte aus dem durch DSKBUFPTR spezifizierten in das Ausgaberegister SEROUT des POKEY geschrieben wird. Ist dieses Zeichen übertragen, wird ein Interrupt ausgelöst.

Weiter wird in dieser Routine nichts getan, als in IRQSTS abzufragen, ob die BREAK-Taste gedrückt wurde. Ist dies der Fall, wird die Übertragung abgebrochen.

Ansonst wird gewartet, bis vom Interrupthandler irgendwann einmal XMITEND auf einen Wert ungleich Null gesetzt wird, da dies das Zeichen für "Übertragung beendet" ist.

Zum Abschluß wird noch der Transmitter über SENDDIS disabled.

$eaad 60077 $ea90 60048 ISRODN

Diese "Interrupt Service Routine if Output Data Needed" wird aufgerufen, wenn der POKEY siganlisiert, er hätte das letzte Zeichen vom Ausgaberegister SEROUT in sein internes Parallel/Seriell-Wandlungsregister übertragen. Dann wird DSKBUFPTR incrementiert und kontrolliert, ob dieser Zeiger sein durch BUFENDPTR angezeigtes Ende erreicht hat. Ist dies der Fall, so wird das Checksummenbyte übertragen, soweit dies noch nicht geschehen und das Flag CHKSUMSND gesetzt ist.

Sind noch Zeichen zu übertragen, wird das nächste Zeichen aus (DSKBUFPTR) genommen und an SEROUT übergeben.

$eaec 60140 $ead1 60113 ISRXD

Signalisiert der POKEY bei einem Interrupt, daß er mit der Übertragung des im Parallel/Seriell-Wandlungsregister enthalten gewesenen Byte fertig und noch kein neues Byte in das SEROUT-Register geschrieben worden sei, wird diese Routine aufgerufen.

Ist hier die Checksumme schon gesendet worden, wird in XMITEND mit einem Wert ungleich Null signalisiert, daß die serielle Ausgabe komplett beendet sei. XMITEND wird nach Beendigung der Interrupt-Routine von SEND abgefragt.

$eafd 60157 $eae2 60130 RECEIVE

Handelt es sich bei dem Lesekommando um eines über die Cassette, wird das Checksummenregister DSKCHECKSUM gelöscht, sonst nicht. Danach werden sowohl bei Cassette, als auch bei jedem intelligenten Gerät die Flags BUFFULL und RECEIVEND gelöscht. Nach dem Aufruf von RECEIVEN, in dem das eigentliche Empfangen erst ermöglicht wird, wird in einer Schleife abgefragt, ob

1) die BREAK-Taste gedrückt ist. Dann wird der Datenempfang abgebrochen,

2) die Timeout-Bedingung über das TIMEFLAG signalisiert wird (dies wäre der Fall, wenn TIMEFLAG den Wert Null hätte)

oder

3) RECEIVEND einen Wert ungleich Null hätte.

In jedem der drei Fälle würde das Lesen beendet, wobei in den ersten zweien dem aufrufenden Modul über DSKSTATUS ein Fehler durchgereicht würde.

$eb2c 60204 $eb11 60177 ISRSIR

Diese "Interrupt Service Routine at Serial Input Ready" wird angesprungen, wenn der POKEY bei einem Interrupt über seine Flags die Meinung kundtut, er hätte ein Zeichen empfangen und es sei in SERIN abzuholen.

In diesem Fall wird der POKEY-Status aus SKSTAT gelesen und eventuelle Fehlerbits durch Schreiben in SKSTATRES gelöscht. Danach wird getestet, ob das angeblich bereitliegende Zeichen richtig gelesen wurde, oder ob ein Framing- oder Overrun-Error aufgetreten ist. In jedem der beiden Fehlerfälle setzt die Routine DSKSTATUS entsprechend.

Ist das Zeichen als richtig übertragen eingezeichnet, wird über BUFFULL geprüft, ob alle einzulesenden Zeichen eingetroffen sind. Ist dies der Fall, ist also BUFFULL nicht Null, wird das Zeichen, das jetzt noch anliegt, als Checksumme interpretiert und nach dem Lesen des Zeichens geprüft, ob sie mit der berechneten Checksumme übereinstimmt. Ist die Prüfsumme nicht in Ordnung, wird in DSKSTATUS ein Checksummenfehler signalisiert. Unabhängig davon wird über das Setzen von RECEIVEND signalisiert, daß RECEIVE terminieren kann.

Ist BUFFUL Null, wird das Zeichen gelesen, zu der Checksumme addiert und in den Puffer ab (DSKBUFPTR) geschrieben. Danach wird DSKBUFPTR incrementiert und geprüft, ob BUFENDPTR erreicht wurde. Ist DSKBUFPTR weiterhin kleiner als BUFENDPTR, wird die Interruptroutine ohne weitere Flagänderungen verlassen.

Bei Gleichheit der beiden Pointer ist das Ende der eigentlichen Datenübertragung gekommen. Es kann nun noch unter Umständen eine Checksumme zu empfangen sein. Ist dies der Fall, also ist NOCHKSUM Null, wird BUFFULL für den nächsten Interrupt auf $ff gesetzt und die Unterbrechungsroutine verlassen.

Ist laut NOCHKSUM keine Prüfsumme mehr von dem Sender zu erwarten, wird NOCHKSUM für den nächsten Aufruf von RECEIVE gelöscht und vor dem Verlassen der Routine RECEIVEND auf $ff (Receive vollständig beendet) gesetzt.

$eb87 60295 $eb6e 60270 LOADPTR

Es werden zwei Berechnungen durchgeführt:

DSKBUFPTR := DSKBUFFER

BUFENDPTR := DSKBUFFER + DSKBYTCNT

Das heißt nichts anderes, als daß die für den SIO benötigeten Bufferanfangs- und Enddaten aus dem DCB in den SIO-Control-Block übertragen werden.

$eb9d 60317 $eb84 60292 CASENTER

Dieser Punkt wird angesprungen, wenn es sich um einen Cassetten-I/O handelt. Dazu wird als erstes unterschieden, ob es sich um einen Lese- oder Schreibzugriff handelt, da beim Lesen die Eingabegeschwindigkeit gemessen wird, beim Schreiben dagegen festliegt.

Handelt es sich also um einen Schreibzugriff, wird der POKEY in seinen Asynchron-Ausgabezählern 3 und 4 auf 600 Bit/Sekunde programmiert. Dann wird über LOADPTR der SIO-Kontrollblock initialisiert und mit SEND der Block gesendet. CASENTER ist ebenfalls zuständig für das Ein- und Ausschalten des Cassettenmotors.

Handelt es sich dagegen um einen Lesezugriff, wird CASFLAG gesetzt und, ebenfalls nach Setzen der SIO-Pufferpointer über LOADPTR, für diesen Block die Lesegeschwindigkeit festgehalten durch Aufruf der Prozedur BEGINREAD. Außerdem wird Timer1 so initialisiert, daß er als Timeout-Interruptgeber fungiert.

Nach einem RECEIVE-Aufruf, in dem der Block entweder richtig gelesen wird oder aber die Flags entsprechend gesetzt werden, wird der Cassettenmotor abgeschaltet, wenn dies gewünscht wurde (nur nach Übertragung des letzten End Of File-Blocks) und zum Aufrufer zurückgekehrt.

$ec11 60433 $ebf0 60400 TIMER1INT

Hier wird hingesprungen, wenn bei einem Vertikal-Blank-Interrupt der TIMCOUNT1 auf Null gegangen ist. Diese Routine wird benutzt, um einen Timeout ('endloses' Warten auf ein Gerät, das nicht antwortet oder nicht antworten kann) zu erkennen und die 'hängende' Operation mit den entsprechenden Fehlermeldungen abzubrechen. Dazu wird einfach eine Null in TIMEFLAG geladen.

Damit TIMER1 diese Routine finden kann, wird in der Prozedur SETTIM1V die Adresse von TIMER1INT nach TIMER1VKT geladen.

$ec17 60439 $ebf6 60406 SENDENABL

Hier wird in SKCNTL und seinem Schattenregister ein Senden über die serielle Schnittstelle initiiert. Wei-

terhin wird bei Cassettenbenutzung für das dabei verwendete Zweitonverfahren Tonregister 2 mit der niedrigen Frequenz und Tonregister 1 mit der hohen Frequenz geladen. Dann werden in IQREN§ eventuelle alte Interrupts gelöscht und der Output_Data_Interrupt enabled. Danach wird mitten in die RECEIVEN-Routine gesprungen, wo die restlichen POKEY-Register für die Übertragung initialisiert werden.

$ec40 60480 $ec1f 60447 RECEIVEN

Hier wird, analog zu SENDENABL, der POKEY auf asynchrone Datenübertragung eingestellt und der Receiver-Interrupt in IRQEN erlaubt.

Außerdem wird SKSTATRES beschrieben, um eventuelle alte Fehlermeldungen zu löschen.

Hiernach steht der Einsprungpunkt für SENDENABL, bei dem Taktkanal 3 mit 1,77MHz und Kanal 4 mit dem Ausgang von Kanal 3 versorgt wird.

Dann werden die entsprechenden Kontrollregister AUDICNTL1 bis AUDICNTL4 entsprechend dem Wert von IOSOUNDEN so gesetzt, daß der "Übertragungs-Sound" nur bei gesetztem IOSOUNDEN durchkommt.

Falls keine Cassettenoperationen gewünscht werden, werden die Tonkanäle 1 und 2 blockiert. Sie werden nur für das Zweitonverfahren der Cassette benötigt.

$ec84 60548 $ec63 60515 SENDDIS

Es werden die Bits 3, 4 und 5 von IRQEN§ gelöscht, was jegliches Lesen und Schreiben über Interrupt unterbindet. Weiter werden die 4 Tonkontrollregister AUDICNTLx gelöscht.

$ec9a 60570 $ec79 60537 SETTIMOUT

Diese Routine liefert das Produkt aus dem Low-Wert von DSKTIMOUT und der Konstanten 32.

Sie liefert den High-Teil des Resultates im X-Register und den Low-Teil im Y-Register zurück.

$eca9 60585 $ec88 60552 INTTAB

Hier liegen die drei Interruptvektoren ISRSIR, ISRODN sowie ISRXD.

$ecaf 60591 $ec8e 60558 SENDINIT

Nach kurzem Delay, das den I/O-Baugruppen Gelegenheit geben soll, sich einzuschwingen, wird SEND aufgerufen und danach WAIT.

Nach Rückkehr von WAIT wird der Y-Wert in den Accu kopiert, um die Flags zu setzen. Ist ein Timeout passiert oder konnte das Gerät die Operation nicht ordentlich durchführen, ist das Zero-Flag gesetzt.

$ecc8 60616 $eca7 60583 COMPUTE

Diese Routine berechnet den Wert für die POKEY-Frequenzregister 3 und 4 beim Cassettenbetrieb. Sie ruft dabei ADJUST auf und wird selbst ausschließlich von BEGINREAD angestoßen.

$ed2e 60718 $ed08 60680 ADJUST

Dieses Unterprogramm wird zur Justage der POKEY-Werte benutzt. Bei Verwendung eines 50Hz-Gerätes wird der Wert des Accus um $20 erhöht, falls er den Wert 124

noch nicht erreicht hat. Ist der Accu größer als $7b, wird der Wert $7c abgezogen.

Bei der NTSC-Version erfolgt die Erhöhung nur um den Wert $07.

$ed3d 60733 $ed14 60692 BEGINREAD

Dieser Programmteil stellt den POKEY auf die hier gemessene Datenübertragungsgeschwindigkeit der Cassette ein. Dazu muß vorausgesetzt werden, daß die beiden ersten Byte eines von Cassette zu lesenden Blockes immer den Wert $aa haben. Dieser Hexadezimalwert entspricht der Binärzahl %10101010, die sich recht gut zur Synchronisation eignet.

Ist die Synchronisation beendet, werden zwei $55- Byte im Datenpuffer abgelegt und die Checksumme dahingehend initialisiert, daß sie mit den beiden $55-Bytes übereinstimmt. Während der gesamten Routine können sowohl Timeout- als auch BREAK-Tasten-Unterbrechungen auftreten. Sie würden durch geeignete Statusmeldungen in DSKSTATUS an das aufrufende Modul zurückgegeben und der Cassettenmotor abgeschaltet werden.

$ede2 60898 $edbd 60861 SETTIM1V

Diese Routine setzt über JUMPTAB+$0c den Sprungvektor für Timer1 (TIMER1VKT) auf TIMOUTINT und über JUMPTAB+$0c den TIMER1 selbst (TIMCOUNT1) auf den in X (HIGH) und Y (LOW) übergebenen Wert.

$edf9 60921 $edd2 60882 POKTAB

Umrechnungstabelle für COMPUTE.

$ee11 60945 $xxxx xxxxx Tabellen

$eeb1 61105 $xxxx xxxxx ADJTAB

In ADJTAB steht der Zeitwert für ADJUST bei der NTSC-Version, in ADJTAB+1 der für die bei uns benutzte PAL-Version.

$eebc 61116 $xxxx xxxxx NEWDEVICE

Diese Routine sucht ab Adresse HATABS nach einem Gerät mit dem in X übergebenen Namen. Findet es diesen Eintrag, so kehrt die Routine mit gesetztem Carry-Flag zurück, wobei das X-Register auf die Handleradrese des gefundenen Eintrags zeigt (also hinter den Namen!) und das Y-Register den Eingangszustand aufweist.

Wird das gesuchte Gerät nicht gefunden, überprüft die Routine noch einmal alle Einträge, ob sie einen leeren Eintrag findet. Ist dies der Fall, wird der übergebene Accu-Wert als HIGH-Teil der Adresse und das Y-Register als LOW-Teil der Handlertabellenadresse interpretiert und zusammen mit dem in X übergebenen Namen als neuer HATABS-Eintrag registriert.

In diesem Fall wird das Carry-Bit gelöscht. Ist kein weiterer Name frei, wird das Y-Register auf $ff und das Carry-Flag auf Eins gesetzt.

$eef9 61177 $xxxx xxxxx SPECHANDL

Dieser SPECial HANDLer ruft DCBINIT mit folgenden Parametern auf:

Accu Ein Zeichen von Adresse (IOCBBUFAZ)

Y Wert von IOCBDSKNZ, also die Laufwerknummer

Kehrt DCBINIT mit negativem Rückgabewert zurück, wird Y mit NON_EXISTANT_DEVICE geladen und die Routine ist beendet.

Anderenfalls werden folgende Register geladen:

Register	geladener Wert	
IOCBCHIDZ	$7f	(8bit)
IOCBPUTBZ	Adr. von PUTBYTE1 - 1	(16bit)
IOCBSPARE+IOCBCHIDZ	Zeich. v. (IOCBBUFAZ)	(8bit)
IOCBSPARE+IOCBCHIDZ+1	CHAINTMP	(8bit)
Y	$01	

$ef26 61222 $xxxx xxxxx PUTBYTE1

Gleich bei Eintritt in diese Routine werden drei mögliche Fehlerquellen überprüft:

1) Die unteren vier Bit des Accu können ungleich Null sein. Dann wird Y mit INVALID_IOCB-NUMBER geladen und die Bearbeitung abgebrochen.

2) Den gleichen Returnwert erhält Y, wenn der Inhalt des X-Registers größer oder gleich $80 ist.

3) Hat STARTTST den Wert Null, wird ein NON_EXISTANT_DEVICE im Y-Register zurückgemeldet.

Liegt keiner der drei Fälle vor, wird der X-Wert als IOCBNUMZ verwendet und der durch IOCBCHIDZ spezifizierte IOCB in den für IOCBs reservierten Zero-Page-Bereich kopiert, um mit diesen neuen Werten PREPLINK aufzurufen. Kehrt diese Routine mit negativem Wert zurück, wird ebenfalls die PUTBYTE-Routine mit einem Y-Wert von FUNCTION_NOT_IMPLEMENTED_IN_HANDLER abgebrochen.

Im anderen Fall wird der Inhalt von IOCBPUTBZ (16 Bit) aus dem Stack abgelegt und durch RTS dieser Vektor angesprungen.

$ef65 61285 $xxxx xxxxx FREI1

Hier liegen sechs unbenutzte und mit Null initialisierte frei zu benutzende Programmspreicherbyte. Bitte bei Belegung auf die korrekte Checksummenberechnung achten (CHECKROM1 u.a.!).

$ef6b 61291 $xxxx xxxxx CASMOTOFFC

Ein direkter Sprung zu CASMOTOFF.

$ef6e 61294 $f3e4 POWERONA

Hier wird LASTCH gelöscht (auf $ff gesetzt), RAMTOP auf den Wert aus RAMSIZE, SHIFTLOCK auf $40 (nur Großbuchstaben), KEYDEFPTR auf KEYDEF und FKTDEFPTR auf FKDEF gesetzt.

$ef8e 61326 $xxxx xxxxx INITSOME

An dieser Stelle wird im Zuge eines Kalt- oder Warmstarts ein großer Teil der im System verwendeten Variablen initialisiert. Dazu gehören zum Beispiel CHARBASE, CHARSTPTR, DMACNTL§, CHARCNTL, MONSTATUS, IRQST§, IRQST, CURSORINH, TEXTINDEX, ADDRESS, TABMAP, COLPF0§..COLBAK§, TEXTSTART und andere.

$f180 61824 $f593 62867 GETCH

Diese Routine liest ein Zeichen ab der angegebenen Cursorposition und übergibt es im Accu. Die Cursorposition wird dabei incrementiert.

$f18f 61839 $f18f 61839 GETPLT

GETPLT ist eine von GETCH benutzte Prozedur und liest

das bei ADRESS liegende Zeichen ein und wandelt es vom internen Bildschirmcode zurück in den ATASCII-Code.

$f1a4 61860 $f5bd 62909 OUTCH

Das im Accu übergebene Zeichen wird auf CLEAR_SCREEN ($7d) und NEWLINE ($9b) überprüft und gegebenenfalls die entsprechenden Routinen aufgerufen, ansonsten wird die Zeichenverwaltung von OUTPLT übernommen, bevor der Cursor incrementiert wird.

$f1ca 61898 $f5e0 62944 OUTPLT

Wenn des Bildschimausgabeflag STARTSTOP nicht Null ist, läuft der Atari hier in einer Endlosschleife so lange, bis es wieder gelöscht wird.

Danach wird das im Accu übergebene Zeichen an der Cursorposition abgelegt.

In dieser Routine ist OUTCH2 ($f1e9) häufig auch von anderen Modulen benutzter Einsprungpunkt.

$f20b 61963 $f621 63009 RETURM

RETURn from Monitor sorgt dafür, daß bei eingeschaltetem Grafikmodus kein Cursor zu sehen ist und versorgt bei erfolgreich verlaufenen I/O DSKSTATUS mit dem Wert SUCCESS.

An dieser Stelle sollte nicht ganz unerwähnt bleiben, daß hier mitten in der Routine (kurz vor dem Ende) umständlich um das folgende Label herumgesprungen werden muß. Wir halten das für dieses recht aufgeräumte Betriebssystem für den Ausrutscher für die Leute, die nicht gerne Perfektes kaufen.

Somit endet die Routine RETURM bei Adresse $f22d und

nicht schon vor dem folgenden Sprung.

Ebenfalls nicht unwesentlich ist, daß die Adresse $f21e von SWAP als Einsprungpunkt benutz wird. Es ist die Stelle, an der der aufrufenden Schicht die Erfolgsmeldung signalisiert wird. Diese Adresse ist beim 400/800 $f634.

$f223 61987 $xxxx xxxxx TESTROMEN

Dies ist lediglich ein direkter Sprung zu SWITCHROM. Bitte vor eventuellen Änderungen vorstehenden Absatz lesen!

$f22e 61998 $xxxx xxxxx SCROLFINE

Ist Bit 7 von FINESCROL Null, wird die Routinenbearbeitung abgebrochen, sonst werden der Display-List-Interrupt abgeschaltet, FINESCROL auf Null und der ANTIC-Programmvektor auf $c0ce, also MASKTAB-1 gesetzt bevor in die Routine INITSOME gesprungen wird.

$f24a 62026 $f63e 63038 EGETCH

Diese Routine wird benutzt, um eine komplette logische Zeile (also über max. 120 Zeichen) einzugeben und edieren zu können. Als Rückgabewert wird im Accu das erste Zeichen der Zeile übergeben, falls nicht BREAK gedrückt wurde. War dies der Fall, so wird im Accu ein NEWLINE ($9b) übergeben und das Y-Register enthält als Status den Wert $80.

War die Eingabe in Ordnung, ist also nicht BREAK gedrückt worden, hat Y den Wert $01. Da beim ersten Aufruf nur das erste Zeichen der eingegebenen Zeile übergeben werden kann, muß nun für jedes weitere eingegebene Zeichen EGETCH erneut aufgerufen werden. Dabei wird dann automatisch immer das nächste Zeichen zurück-

geliefert. Das Ende einer logischen Zeile wird dem aufrufenden Modul ebenfallsn mit dem NEWLINE-Kode mitgeteilt.

$f2ad 62125 $f6a1 63137 JSRIND

Es erfolgt von hier aus ein indirekter Sprung zu (ADDRESS).

$f2b0 62128 $f6a4 63140 EOUTCH

Es wird das im Accu übergebene Zeichen an der Cursorposition abgelegt. Steuerzeichen werden verarbeitet.

$F2f8 62200 $f6dd 63197 KGETC2

Dies ist kein Einsprungpunkt, das Label wird lediglich von der folgenden KGETCH-Routine benötigt.

$f302 62210 $f6e2 63202 KBGETCHAR

Hier wird ein Zeichen von der Tastatur gelesen und auf verschiedene Sonderzeichen hin überprüft. Eines dieser Sonderzeichen wäre zum Beispiel das mit dem Tastaturcode $89. Dieses leider im 600XL/800XL nicht verfügbare Sonderzeichen würde den Tastaturklick toggeln, das heißt, einschalten, wenn er ausgeschaltet war und umgekeht. Weiterhin behandelt die Routine aber auch beim 600XL/800XL zu verwendende Sonderzeichen wie zum Beispiel CONTROL-1 zum Anhalten und Weiterlaufenlassen der Bildschirmausgabe.

$f3e0 62432 $f779 63353 ESCAPE

Das ESCAPEFLG wird auf $80 gesetzt. Es wird benötigt, um Cursorbefehle als Sonderzeichen auf dem Bildschirm

darstellen zu können.

$f3e6 62438 $f77f 63359 CURSORUP

Der Cursor wird um eine physikalisch Zeile nach oben transportiert beziehungsweise am oberen Bildschirmrand auf die unterste Zeile zurückgesetzt (sogenanntes wrap-around).

$f3f3 62451 $f78c 63372 CURSORDWN

Der Cursor wird eine Zeile nach unten beziehungsweise von der letzten Zeile aus wieder in die oberste zurückgesetzt.

$f400 62464 $f799 63385 CURSORLFT

Der Cursor wird eine Position nach links geschoben. Befindet er sich zu Beginn am Anfang einer Zeile, so erfolgt ein Zeilen-wrap-around.

$f411 62481 $f7aa 63402 CURSORRIG

Der Cursor wird eine Spalte nach links weitergeschoben. Befindet er sich am äußersten Rand des Bildschirms, wird er auf die erste Spalte derselben Zeile gesetzt.

$f420 62496 $f7b9 63417 CLEARSCRN

Der Bildschirm wird zusammen mit der LOGIGMAP gelöscht.

Danach wird die CURSORHOM-Funktion ausgeführt.

$fe40 62528 $f7d6 63446 CURSORHOM

Der Cursor wird in die linke obere Ecke (Home-Position) gesetzt. Der Bildschirminhalt bleibt unverändert.

$f450 62544 $f7e6 63462 CURSORBS

Es wird die BACK-SPACE-Funktion ausgeführt, d.h., das direkt links vom Cursor stehende Zeichen wird gelöscht und der Cursor steht nun auf dessen Platz.

Steht der Cursor zu Beginn der Routine am Anfang der Zeile, wird das Kommando ignoriert.

$f47a 62586 $f810 63504 CURSORTAB

Es wird in BITMASK nach der nächsten Tabulatorposition gesucht und diese angesprungen.

Ist keine weitere Tabulatorposition in der Zeile verfügbar, wird an den Anfang der nächsten Zeile gesprungen.

$f495 62613 $f82d 63533 CURSOSTAB

Die Spalte, an der der Cursor bei Aufruf der Routine steht, wird Tabulatorposition durch Setzen des entsprechenden Bits in BITMASK.

$f49a 62618 $f832 63538 CURSOCTAB

Eine eventuell an der Cursorspalte existente Tabulatorposition wird gelöscht.

$f49f 62623 $f837 63543 INSCHAR

An der Cursorposition wird nach Möglichkeit ein Leerzeichen eingeschoben (Insert).

$f4d5 62677 $f86d 63597 DELCHAR

Das Zeichen, das logisch rechts vom Cursor steht, wird gelöscht und alle weiteren Zeichen werden 'herangerückt' (Delete).

$f50c 62732 $f8a5 63653 INSLINE

An der Cursorposition wird eine neue logische Zeile eingefügt.

$f520 62752 $f8d4 63700 DELLINE

Die durch die Cursorposition bezeichnete logische Zeile wird gelöscht. Alle anschließenden Zeilen rücken auf.

$f556 62806 $f90a 63754 BELL

Es wird 32 mal KEYCLICK aufgerufen. Das ergibt einen ca. 0,25s langen Ton.

$f55f 62815 $xxxx xxxxx BOTTOMLIN

Diese Routine positioniert den Cursor in die unterste Bildschirmzeile und erste Spalte. Diese Position wird von manchen Terminals als Home-Position angesprungen und ist immer dann sinnvoll einzusetzen, wenn man nicht genau weiß, wo man eigentlich auf dem Schirm steht und was für Informationen auf ihm stehen.

$f565 62821 $f917 63767 DBDDEC

ADDRESS wird zweifach decrementiert und überprüft, ob das Register dann immer noch in den ihm vorgegebenen Grenzen liegt oder ein SCREEN_ERROR aufgetreten ist.

$f5ac 62892 $f947 63815 CONVERT

Die durch Spalten- und Zeilennummer spezifizierte Cursoradresse wird in die effektive Speicheradresse umgerechnet.

$f60a 62986 $f9d4 63956 INCRSB

Nach einem Incrementieren der Cursorposition wird überprüft, ob der Cursor am Ende einer Zeile oder ab Bildschirmende steht.

Sollte dies der Fall sein, wird automatisch ein Scroll-Vorgang ausgelöst.

$f6ae 63150 $fa7a 64122 SUBEND

Je nach Wert von X ($00 oder $02) wird ENDPOINTR von PLOTROWAC oder von PLOTCOLAC abgezogen.

$f6bc 63164 $fa88 64136 ERANGE

ERANGE ist der Einsprungpunkt für den Editor. Hier wird getestet, ob der Editor geöffnet ist und bei negativem Ergebnis ein Öffnen vollzogen.

Danach wird überprüft, ob der Cursor innerhalb seiner normalen Grenzen liegt. Tut er dies nicht, wird CURSORHOM aufgerufen und ein CURSOR_OVERRUN-Error signalisiert und die letzte Returnadresse vom Stack genommen. Damit gelangt der Fehler also direkt zu dem

CIOMAIN aufrufenden Modul.

$f715 63253 $xxxx xxxxx JMPF21E

Es folgt ein direkter Sprung mitten in die RETURM-Routine. Dieser Ansprung wird von SWAP benutzt.

$f718 63256 $fae4 64228 OFFCURSOR

Das Zeichen, an dem der Cursor im Moment steht, wird wieder in den Bildschirm geschrieben - der Cursor also abgeschaltet.

Alle jetzt folgenden Routinen mit Namen BITxxx beziehen sich auf Tabulatorenverarbeitung:

$f723 63267 $faeb 64235 BITCON

Die Maske ab MASKTAB+Offset im Accu wird in BITMASK abgelegt und in X 1/8 des im Accu übergebenen Offsets zurückgegeben.

$f732 63282 $fafa 64250 BITROL

LOGICMAP bis LOGICMAP+2 wird um ein Bit nach links geschoben. Das höchste Bit aus LOGICMAP wird im Carry zurückgegeben.

$f73c 63292 $fb04 64260 BITPUT

Setze durch Accu spezifiziertes Bit in TABMAP.

$f74a 63306 $fb12 64274 BITCLR

Löscht das durch den Accu spezifizierte Bit in TABMAP.

$f758 63320 $fb20 64288 LOGGET

Lädt Accu mit CURSROW + 120 und läuft in die nächste Routine hinein:

$f75d 63325 $fb25 64293 BITGET

Gibt ein gesetztes Carry-Flag zurück, wenn das durch den Accu spezifizierte Bit gesetzt ist.

$f76a 63338 $fb32 64306 INATAC

Das im Accu übergebene Bildschirmkode-Zeichen wird in ein ATASCII-Zeichen gewandelt, wenn es sich nicht um ein Grafiksymbol handelt.

$f78e 63374 $fb4e 64334 LINEINSERT

Hier erfolgt das hardwareabhängige Verschieben einer physikalischen Zeile. Es werden die Register ADDRESS, MLTTMP sowie SAVEADR benutzt.

$f7c2 63426 $fb7b 64379 EXTEND

Diese Routine verlängert eine logische Zeile um eine weitere physikalische.

$f7e2 63458 $fb9b 64411 CLEARLINE

Diese Routine erledigt das hardwareabhängige Löschen (nicht: Entfernen!) einer physikalischen Zeile.

$f7f7 63479 $xxxx xxxxx DOSCROLL

Diese Routine erledigt das 'feine' Scrolling auf dem Bildschirm sowie einen Teil der Grafikverarbeitung.

$f8b1 63665 $fc00 64512 DOBUFC

Diese Routine berechnet die Pufferlänge der momentanen logischen Zeile, wobei die letzten Leerzeichen ignoriert werden.

$f90c 63756 $fc5c 64604 STRBEG

BUFSTR wird auf den Anfang der durch die Cursorposition bestimmten Zeile gesetzt.

$f918 63768 $fc68 64616 DELTIA

Diese Routine hat die Aufgabe, eine physikalische Zeile dann zu löschen, wenn sie leer und letzte physikalische einer logischen Zeile ist.

$f93c 63804 $fc8d 64653 TESTCNTL

Diese Prozedur durchsucht die Kontrollzeichentabelle ab CONTROLS. Ist das Zeichen gefunden, hat X den Offset auf den gefundenen Tabelleneintrag bezüglich CONTROLS. Außerdem ist im Erfolgsfall das Zero-Flag gesetzt.

$f94c 63820 $fc9d 64669 PHACRS

CURSROW, CURSCOL und GRAPHEMUL werden ab TEMPROW nach unten abgelegt.

$f957 63831 $fca8 64680 PLACRS

Die von PHACRS ab TEMPROW nach unten hin geretteten Cursorpositionen werden wieder geladen.

$f962 63842 $fcb3 64691 SWAP

Es werden die Variablen CURSROW bis OLDGRADR+1 mit TEXTROW bis TEXTGRAD+1 vertauscht und SWAPFLAG invertiert. Diese Vertauschung muß vorgenommen werden, wenn von Text nach Grafik innerhalb eines Bildschirmes umgeschaltet werden soll oder umgekehrt.

$f983 63875 $fcd8 64728 KEYCLICK

An dieser Stelle wird das Klicken der Tastatur generiert.

$f997 63895 $fce4 64740 COLCR

CURSCOL wird Null, wenn SWAÜFLAG Null und GRAPHEMUL ungleich Null ist. Ansonsten wird CURSCOL auf den Wert von LFTMARGIN gesetzt.

$f9a6 63910 $fcf3 64755 PUTMSC

Der Wert von SCRNSTART wird nach ADDRESS geladen.

$f9af 63919 $fcfc 64764 DRAWTO

Es wird, je nach Kommando in IOCBCMDZ entweder eine Linie gezogen (DRAW) oder ein Bereich ausgefüllt (FILL).

Die Linie würde gezogen von OLDGRROW, OLDGRCOL zu der in CURSROW und CURSCOL angegebenen Adresse.

$fb04 64260 $fe45 65093 TABELLEN

$fb0d 64269 $fec6 65222 CONTROLS

Diese Tabelle beinhaltet alle benutzbaren Steuerzeichen und dahinter die entsprechende Handleradresse.

$fb11 64273 $xxxx xxxxx FKDEF

Hier stehen für jede (beim 600XL/800XL nicht eingebaute) Funktionstaste F1..F4 16 Byte zu ihrer Bedeutungsdefinition zur Verfügung.

$fb51 64337 $xxxx xxxxx KEYDEF

Hier stehen alle 192 möglichen Tastenkodes, respektive ihre Bedeutungen.

$fc1a 64538 $ffbe 65470 CPIRQQ

Hier liegt der Keyboard-Interrupt. Es wird das STARTSTOP-Flag bei Drücken von CONTROL-1 modifiziert, das ATRACT-Register wird gelöscht und SRTIMER auf den vorgewählten Delaywert KEYRPDELY gesetzt.

$fcd6 64726 $xxxx xxxxx FREI2

Es stehen hier 2 mit $00 initialisierte Bytes.

$fcd8 64728 $xxxx xxxxx KEYCLICKC

Von hier wird ein direkter Sprung zur KEYCLICK-Routine gestartet.

$fcdb 64731 $ef41 61249 INIT

In dieser kurzen Routine wird der POKEY auf 600 Bit/Sekunde Datenübetragungsrate eingestellt.

$fce6 64742 $ef4c 61260 CASOPEN

Hier wird der Cassettenrecorder für die Ausgabe oder Eingabe vorbereitet. Es wird ein Fehler zurückgemeldet, wenn das Gerät schon geöffnet ist.

Tritt hier kein Fehler auf, wird je nach Kommando (Schreiben oder Lesen) BEEPWAIT für einen oder zwei Töne aufgerufen, was wiederum einen, beziehungsweise zwei Kommandotöne zur Cassettenbedienung erzeugt. Danach wird nach Drücken einer Taste der Motor hardwaremäßig eingeschaltet und kurze Zeit gewartet, bis er richtig läuft. Beim Schreiben wird gleich ein 20 Sekunden langer Startton an die Cassette gesendet und es werden allgemein die Pufferpointer und Pufferlängenzeiger gesetzt.

$fd7a 64890 $efd6 61398 CASRDBYTE

Es wird ein Byte von Cassette gelesen. Ist der Puffer leer, aber das File noch nicht zu Ende, so wird ein neuer Datenblock eingelesen. Das Datum wird bei richtigem Lesen im Accu zurückgegeben und in Y ergibt sich der Status der Operation.

$fd8d 64909 $efe9 61417 CASREADBL

Es wird ein kompletter Block von Cassette gelesen und kontrolliert, ob es der letzte Block war. Ist dies der Fall, wird weiter kontrolliert, wie viele Byte noch in diesem Block enthalten sind. Die Anzahl wird in CASBUFLIM zurückgegeben.

War das File schon komplett gelesen, erfolgt END_OF_FILE-Errormessage.

$fdb4 64948 $f010 61456 CASPUTBYT

Das im Accu übergebene Zeichen wird im Puffer ab CASDATA bis CASDATA+$7f abgelegt. Ist der Puffer voll (CASBUFPTR zeigt in den Puffer hinein auf die nächste freie Stelle), wird er automatisch geschrieben und neu initialisiert.

$fdcc 64972 $f028 61480 STATUS

Hier wird lediglich eine Erfolgsmeldung (SUCCESS) in Y signalisiert.

$fdcf 64975 $f02b 61483 CASCLOSE

Es wird die Cassettenbearbeitung abgeschlossen. Beim Lesen ist dies recht einfach, beim Schreiben muß jedoch darauf geachtet werden, daß die letzten im Puffer vorhandenen Bytes ebenfalls auf die Cassette geschrieben werden. Zusätzlich muß nach dem letzten Block noch ein End_Of_Text-Block geschrieben werden. Erst danach darf der Cassettenmotor abgeschaltet werden.

$fdfc 65020 $f058 61528 BEEPWAIT

Im Accu wird an diese Routine die Anzahl der Piep-Töne übergeben, die ausgesendet werden sollen. Jeder der Töne ist ca. 0,25s lang, danach folgt eine Pause von ca. 0,1s. Nach dem letzten Ton wird KBGETCHAR aufgerufen, also auf die Eingabe eines beliebigen Zeichens gewartet.

$fe3f 65087 $f095 61589 SIOSYSBUF

Hier wird der SIO-Puffer ab DSKDEVICE für die Cassettenoperationen Read beziehungsweise Write vorbereitet. Zum Beispiel wird der Cassettenpuffer auf $03fd gelegt und die Anzahl der zu erwartenden beziehungsweise zu schreibenden Zeichen auf 131. Dann wird über JUMPTAB+$09 das SIOINTERFace aufgerufen.

Der jeweilige Kommandokode ist im Accu zu übergeben. Wird SIOSYSBUF mit einem Write-Kommando aufgerufen, ist die Routine höchstwahrscheinlich von WSIOSB angesprochen worden.

$fe7c 65148 $f0d2 61650 WSIOSB

Dies ist die Vorbereitungsroutine für das Schreiben auf Cassette. Im Accu wird der Typ des zu schreibenden Blockes übergeben. Dabei bedeutet

$fc Dieser Block besteht aus 128 Datenbyte

$fa Dieser Datenblock enthält weniger als 128 signifikante Byte. Die genaue Anzahl steht in Byte 128 des Blockes. Das heißt, es werden zwar 128 Byte gelesen, aber nur maximal 127 verwendet.

$fe Es handelt sich hierbei um einen END_OF_FILE-Block, bei dem alle Daten-

byte initiell Null sind.

Es muß an diese Routine kein Kommando übergeben werden. Sie wird nur im Scheib-Fall aufgerufen.

$fe8d 65165 $xxxx xxxxx TABELLE

Sie enthält die Wertepaare

$04	$03
$80	$c0
$02	$01
$40	$e0
$1e	$19
$0a	$08

$fe99 65177 $ee78 61048 PHINIT

Der Wert für den Timeout beim Printer (PTIMOUT) wird auf 30 gesetzt.

$fe9f 65183 $ee7e 61054 PHSTLO

Hier liegt für die indirekte Benutzung der Adreßwert $02ea (DEVICSTAT).

$fea1 65185 $ee80 61056 PHCHLO

Hier liegt für die indirekte Benutzung der Adreßwert $03c0 (PRINTBUF).

$fea3 65187 $ee81 61057 PHSTAT

Es wird versucht, den Drucker anzusprechen. Wenn dies

nicht gelingt, wird ein gesetztes Negativ-Flag in der CPU zurückgeliefert.

$fec2 65218 $ee9f 61087 PHOPEN

Nach einem Aufruf von PHSTAT wird die Printerpufferlänge auf 0 gesetzt (PRTBUFPTR). Der Status aus PHSTAT liegt noch im Y-Register, die Flags sind nicht signifikant.

$fecb 65212 $eea7 61095 PHWRITE

Der Printerpuffer PRINTBUF wird successive gefüllt bis zu einem NEWLINE-Zeichen. Ist dann der Puffer noch nicht voll (128 Zeichen), wird er mit Leerzeichen aufgefüllt.

Ist der Puffer voll, wird er über JUMPTAB+$09 und SIOINTERF auf den Drucker geschickt.

$ff02 65282 $eedc 61148 PHCLOSE

Nach einem Aufruf von PRMODE wird geprüft, ob der PRINTBUF leer ist und wenn nicht, wird er über PHWRITE geleert. PHWRITE wird nicht am eigentlichen Einsprungpunkt aufgerufen.

$ff0f 65295 $eee6 61158 SETDBC

In X und Y werden der Low- beziehungsweise High-Teil der Pufferadresse aus PHCHLO übergeben, um damit und mit weiteren intern zu holenden Werten die SIO für einen Druck-Befehl zu initialisieren.

$ff3f 65343 $ef1a 61210 PHPUT

Der Wert aus PRTTIME wird nach PTIMOUT übertragen.

$ff46 65350 $ef1a 61210 PRMODE

Je nach Printmode (Normal, Doppelte_Breite oder Schmaldruck) werden DSKCMD und DSKAUX1 initialisiert.

$ff6e 65390 $xxxx xxxxx CHECKROM1

Es wird die Checksumme über die folgenden Speicherbereiche berechnet:

$c002	...	$cfff	(c000,1 nicht, weil sie selbst eine Checksumme darstellen und deshalb nicht mitberechnet werden können!
$5000	...	$57ff	(TestROM)
$d800	...	$dfff	(Mathe-ROM)

Ist diese in CHECKSUM (LOW) und CHECKSUM+1 (HIGH) berechnete Prüfsumme identisch mit den in CHECKSR0 beziehungsweise CHECKSR1 programmierten Werten, erfolgt positive Meldung durch gelöschtes Carry-Flag, sonst ist das Carry-Flag gesetzt. Die Routine kann natürlich auch zum Berechnen der Werte bei neu programmierten Betriebssystemteilen verwendet werden. Die Werte stehen nach der Berechnung ja immer noch in CHECKSUM zur Verfügung!

Diese und die nächste Funktion benutzen GETCHECKS.

$ff8d 65421 $xxxx xxxxx CHECKROM2

Diese Funktion liefert als Wert ein gelöschtes Carry-Flag, wenn die in GETCHECKS für die ROM- (RAM-?) Bereiche

$e000 ... $fff7 ($fff8, $fff9 ist Prüfsumme)
$fffa ... $ffff

berechnete Prüfsumme mit den Werten in CHECKSUM2 (Low) und CHECKSUM2+1 (High) übereinstimmt, ansonsten ist der Funktionswert im Carry-Flag Eins.

$ffa4 65444 $xxxx xxxxx GETCHECKS

Bei Aufruf dieser Routine erwartet diese im X-Register einen Wert, aus dem sie den Speicherbereich erkennen kann, dessen Checksumme sie zu dem Wert in CHECKSUM (Low) und CHECKSUM+1 (High) addieren kann, wobei es egal ist, ob dieser Speicherbereich RAM oder ROM selektiert. Es stehen folgende Werte für X zur Verfügung, die dann über CHKSUMTAB die entsprechenden Speicherbereiche spezifizieren:

Eingabeparameter in X	geprüfter Speicherbereich
$00	$c002 ... $cfff
$04	$5000 ... $57ff
$08	$d800 ... $dfff
$0c	$e000 ... $fff7
$10	$fffa ... $ffff

Die Routine kehrt als Seiteneffekt mit einem um den Wert 4 erhöhten X-Register zurück. Das erleichtert das mehrfache Aufrufen dieser Funktion zum Testen mehrerer Speicherbereiche.

$ffd2 65490 $xxxx xxxxx CHKSUMTAB

Hier stehen die Anfangs- und um eins erhöhten Endwerte für den Prüfsummentest GETCHECKS. Es sind demzufolge die Werte

$c002	$d000
$5000	$5800
$d800	$e000
$e000	$fff8
$fffa	$0000

enthalten.

$fff8 65528 $xxxx xxxxx CHECKSUM2

Hier und im folgenden Byte steht die Prüfsumme über die momentan implementierten Betriebssystemteile in den Speicherbereichen

$e000 ... $fff7
$fffa ... $ffff

Die beiden Prüfsummenbytes $fff8 und $fff9 sind selbstverständlich nicht enthalten. Dies wäre zwar rechnerisch möglich, würde jedoch den Nachteil in sich bürgen, daß die Funktion GETCHECKS nicht mehr als Seiteneffekt bei Neuimplementierungen von Betriebssystemteilen die neue, korrigierte Prüfsumme liefern würde.

$fffa 65530 $fffa 65530 NMIVKT

Dieser Vektor ist der vom Mikroprogramm der R6502-CPU festgelegte Vektor für die Routine des Nichtmaskierbaren Interrupts. Er zeigt auf die Adresse PNMI.

$fffc 65532 $fffc 65532 RESETVKT

Dieser Vektor ist der vom Mikroprogramm der CPU festgelegte Vektor für die Routine des Hardware-Reset. Er zeigt auf die Adresse RESETCOLD.

$fffe 65534 $fffe 65534 INTVKT

Dieser Vektor ist der vom Mikroprogramm der CPU festgelegte Vektor für die Routine des maskierbaren Interrupt. Er zeigt auf die Adresse JMPIRQVKT.

D E R   S P E I C H E R P L A N   D E S   A T A R I

0 $0 HILFSWORT
1 $1

Diese zwei Byte werden von der Reset-Routine beim Speichertest verwendet.

2 $2 CASINITV
3 $3

Wurde von Cassette gebootet, und war das Booten erfolgreich, erfolgt ein Sprung an die hier stehende Adresse.

4 $4 RAMTSTPTR
5 $5

Auch dieser Pointer wird nur für den Speichergrößentest verwendet.

6 $6 TMPRAMSIZ

Wird in Verbindung mit Adresse 5 ($5) benutzt, um das Ergebnis des RAM-Tests festzuhalten.

7 $7 TESTDATA

Hier steht das Datenbyte, das vor Beginn des Speichertests an der gerade zu testenden Stelle stand.

8 $8 WARMFLAG

Steht hier ein Wert ungleich Null, erfolgt im Normalfall beim Drücken der RESET-Taste nur ein Warmstart.

9	$9	DOSAKTIV

Hat dieses Byte den Wert Eins, so erfolgt beim Warmstart ein Sprung zur DOS-Initialisierungsroutine in DOSINIT.

10 11	$a $b	DOSVKT

Hier liegt die Startadresse der DOS- oder anderen Boot-Software.

12 13	$c $d	DOSINIT

Sprungadresse zur Initialisierungsroutine des DOS.

14 15	$e $f	BASMEMTOP

Hier steht die höchste, vom Benutzerprogramm zu verwendende Speicheradresse. Darüber liegt in den meisten Fällen der Bildschirmbereich.

16	$10	IRQEN$

Durch Setzen der jeweiligen Bits können hier die Interruptquellen vom POKEY gesteuert werden. Ist ein Bit gesetzt, so ist die entsprechende Quelle aktiv.

17	$11	IRQST$

Ist eines der Bits dieses Schattenregisters auf Null, so ist die zugehörige Interruptquelle im POKEY aktiv geworden und es ist der entsprechende Handler aufzurufen.

18 $12 CLOCK
19 $13
20 $14

Dieser Drei-Byte-Wert wird alle 1/50 Sekunde incrementiert, wobei 20 ($14) das niederwertigste Byte ist.

21 $15 BUFFERADR
22 $16

Diese Adresse dient den SIO-Routinen als Hilfszeiger bei Diskettenoperationen.

23 $17 IOCBCMD

Sie dient als Hilfsspeicher bei CIO-Operationen.

24 $18 DISKFORM
25 $19

Dies ist ebenfalls ein Hilfszeiger für die Diskettenoperationen.

26 $1a DISKUTIL
27 $1b

Für diesen Pointer gilt ähnliches wie für DISKFORM.

28 $1c ABUFPTR0
29 $1d ABUFPTR1
30 $1e ABUFPTR2
31 $1f ABUFPTR3

Dies sind Hilfszeiger für rein internen Gebrauch.

32 $20 IOCBCHIDZ

In diesem Byte liegt das Erkennungssymbol des anzusprechenden Gerätes an der seriellen Schnittstelle.

33 $21 IOCBDSKNZ

Bei der Diskettenverarbeitung reicht die Identifikation 'Diskette' in IOCBCHIDZ nicht aus, so daß noch eine Laufwerksnummer übergeben werden muß. Dies geschieht hierdurch.

34 $22 IOCBCMDZ

Hier liegt das gerade in Arbeit befindliche oder gerade fertig gewordene CIO-Kommando.

35 $23 IOCBSTATZ

An dieser Adresse legt die CIO-Routine ihre Statusmeldungen ab.

36 $24 IOCBBUFAZ
37 $25

Hier liegt die Anfangsadresse des Datenpuffers bei CIO-Operationen.

38 $26 IOCBPUTBZ
39 $27

Startadresse-1 der PUT_ONE_BYTE-Routine des entsprechenden Gerätes.

40 $28 IOCBBUFLZ

41 $29

Pufferlänge ab IOCBBUFAZ

42 $2a IOCBAUX1
43 $2b IOCBAUX2
44 $2c IOCBAUX3
45 $2d IOCBAUX4

Hilfsinformationen für die CIO-Kommandobearbeitung.

46 $2e IOCBNUMZ

Nummer des zu benutzenden IOCBs, multipliziert mit 16.

47 $2f IOCBCHARZ

Hilfsregister zur Aufnahme des zu übertragenden Zeichens bei CIO-Schreib-Operationen ohne Datenpuffer.

Die folgenden Register sind ausschließlich für den internen Gebrauch bestimmt. Sie dürfen nur benutzt werden, wenn die sie benutzenden Routinen verändert werden, da sonst nichts über den Zustand sowohl der benutzenden Routinen als auch der Variablen selbst ausgesagt werden kann.

48 $30 DSKSTAT

Statusrückmeldung aus Disketten- oder Cassettenverarbeitung.

49 $31 DSKCHKSUM

Prüfsummenregister für serielle Übertragung.

50 $32 DSKBUFPTR
51 $33

Pufferanfang für Cassetten oder Disketten-operation.

52 $34 BUFENDPTR
53 $35

Zeigt auf das Ende des Disk-Puffers BUFENDPTR.

54 $36 LOADERTMP
55 $37

Hilfsregister für den internen Gebrauch des Loaders.

56 $38 BUFFULL

Ist dieses Flag ungleich Null, so ist der CIO- beziehungsweise SIO-Puffer voll.

57 $39 RECEIVEND

Ist dieses Flag nicht Null, so bedeutet es, daß die CIO-Routine ihren Empfang beendet hat.

58 $3a XMITEND

Dieses Flag gibt darüber Auskunft, ob die interruptgesteuerte Senderoutine ihre Aufgabe schon beendet hat.

59 $3b CHKSUMSND

Wenn gesetzt, ist die Checksumme bereits gesendet.

60 $3c NOCHKSUM

Bei einem Wert ungleich Null wird keine Checksumme gesendet.

61 $3d CASBUFPTR

Bytezähler für die Cassettenübertragung. Liegt wertmäßig zwischen Null und CASBUFLIM.

62 $3e GAPTYPE

$00: Normale Gaplänge zwischen den einzelnen Blöcken auf einer Cassette.
$80: Extrem langer Gap am Anfang der Cassettenübertragung.

63 $3f CASEOF

Ist dieses Byte gesetzt, so hat die Leseroutine der Cassette einen END_OF_FILE-Record gefunden.

64 $40 BEEPCOUNT

Hier steht der Parameter für die BEEPWAIT-Routine. Er enthält die Anzahl der abzugebenden Huptöne als Erkennung für irgendwelche externen Operationen (z.B. zwei Huptöne: Schalte PLAY und RECORD ein).

65 $41 IOSOUNDEN

Wenn dieses Byte Null ist, wird die Tonausgabe der seriellen Ein-/Ausgabe unterdrückt.

66 $42 CRITICIO

Dieses Bit wird dann gesetzt, wenn eine hohe, schnelle Folge von Interrupts erwartet wird (z.B. bei seriellem I/O). Sie bewirkt, daß die Vertical-Blank-Unterbrechungsroutine nur zu einem sehr geringen Teil bearbeitet wird, um somit Zeit zu sparen. Als Seiteneffekt ergibt sich zum Beispiel, daß aufgrund der fehlenden Zählerergebnisse der Autorepeat der Tastatur unwirksam ist.

67 $43 FILEMNGMT
bis
73 $49

Interne Pointer für Diskettensteuerung.

74 $4a ZCHAIN
75 $4b

Pointer für die Verarbeitung der linearen Liste der IOCBs.

76 $4c MONSTATUS

Dieser Status wird von der Monitorverarbeitung benötigt.

77 $4d ATTRACT

Ist dieses Byte niedriger als 128 = $80, so erfolgt normale Bildschirmausgabe. Erreicht es den Wert 128, wird es auf 254 gesetzt und bwirkt das Einschalten des sogenannten Attract-Modes.

78 $4e ATRACTMSK

$fe: Normale Bildschirmhelligkeit
$f6: verringerte Helligkeit

ATRACTMASK ist abhängig von ATTRACT.

79 $4f COLREGSH

Dieses Register gehört ebenso wie ATRACTMSK zur Verarbeitung des Attract-Modes. Es erfolgt ein logische Verknüpfung der Farb- und Luminanzregister des ANTIC.

80 $50 MONTEMP
81 $51

Hilfsbyte für Bildverarbeitung.

82 $52 LFTMARGIN

Wert des linken Randes bei Textdarstellung.

83 $53 GIGMARGIN

Wert des rechten Randes bei Textdarstellung.

84 $54 CURSROW

Dies ist die momentane Cursorreihe bei Grafikverarbeitung in dem Bereich von 0 bis 191.

85 $55 CORSCOL
86 $56

Cursorspalte bei Grafikverarbeitung in den Bereichen von 0 bis 319.

87 $57 GRAPHEMUL

Dieser Wert legt fest, in welchem Grafik-Modus die folgenden Ausgaben stattfinden. Es wird benötigt, um von höheren Programmiersprachen aus leicht mehrere Grafikarten mischen zu können.

88 $58 SCRNSTART
89 $59

Adresse des ersten Bildschirmbytes.

90 $5a OLDGRROW
91 $5b OLDGRCOL
92 $5c
93 $5d OLDGRCHR
94 $5e OLDGRADR
95 $5f

Diese Adressen beinhalten alle Daten, die zwischengespeichert werden müssen, wenn zwischen Grafik- und Textverarbeitung während eines Programms umgeschaltet werden soll. Auch diese Adressen sind nur für interne Zwecke bestimmt!

96 $60 FKTDEFPTR
97 $61

Hier steht die Anfangsadresse der 8 Byte langen Tabelle zur Festlegung der Funktionstastenkodes beim 1200XL.

98 $62 PALNTSC

0: PAL-Fernsehsystem
1: NTSC

99 $63 AKTCHRNUM

Diese Variable enthält die aktive Cursorposition innerhalb einer logischen Zeile.

100 $64 ADDRESS
101 $65
102 $66 MLTTMP
103 $67
104 $68 SAVEADR
105 $69

Recht häufig intern benutzte Zwischenspeicher für vielfältige Aufgaben.

106 $6a RAMTOP

Dieses Register hält die Anzahl der im Computer insgesamt zur Verfügung stehenden RAM-Pages.

107 $6b AKTBUFLEN

Hier steht die momentane Größe der aktuellen logischen Zeile.

108 $6c BUFSTR
109 $6d

GETCHARACTER-Pointer des Editors.

110 $6e BITMASK

Register für die Verwaltung logischer Zeilen.

111 $6f SHFAMT

Ebenfalls für GETCHAR benötigtes Hilfsregister.

112 $70 PLOTROWAC
113 $71
114 $72 PLOTCOLAC
115 $73
116 $74 ENDPOINTR
117 $75
118 $76 DELTAROW
119 $77 DELTACOL
120 $78

Diese Register werden zusammen mit den zwei weiter hinten stehenden Registern ROWINC und COLINC zur Berechnung von Grafik benutzt (z.B. DRAWTO).

121 $79 KEYDEFPTR
122 $7a

Dieser Pointer zeigt auf die Tabelle zur Wandlung von Tastaturkode nach ATASCII.

123 $7b SWAPFLAG

Ungleich Null zeigt dieses Flag an, daß der Bildschirmvariablenbereich auf Grafik eingestellt ist, Null signalisiert, daß die Variablen für den Bildschirmbereich Referenzen auf Texte darstellen.

124 $7c HOLDCHAR
125 $7d INSDATA
126 $7e COUNTER
127 $7f

Ebenfalls alles interne Hilfsvariable für die Verwaltung des Bildschirms.

128 $80 LOWMEM
129 $81

Dieser Zeiger weist auf die unterste Adresse, die nicht mehr für Betriebssystemzwecke verwendet wird.

139 $8b CHECKSUM
140 $8c

In diesen zwei Byte wird die Prüfsumme über Speicherbereiche gebildet. Da diese Checksumme nur bei der Initialisierungsphase benötigt wird, kann der Wert nach dem Kaltstart zerstört werden.

512 $200 DLIVKT
513 $201

Vektor für die ANTIC-Programm- Unterbrechung. Wenn eine ANTIC-Programm-Unterbrechung ausgelöst wird, wird ein Unterbrechungsprogramm, dessen Adresse in diesen beiden Speicherstellen steht, ausgeführt.

514 $202 VPRECEDE
515 $203

Ansprungvektor für von PORT A ausgelösten Interrupt, der eine SERIAL-BUS-PROCEED-Unterbrechung darstellt.

516 $204 VINTERRUP
517 $205

Der Interrupt von PORT B signalisiert eine Unterbrechungsanforderung für die serielle Übertragungsunterbrechung. Die letzten beiden Interruptvektoren werden normalerweise vom System ignoriert, da die angeschlossenen Geräte keine Interrupts erzeugen. Die Leitungen sind für spätere Erweiterungen eingeplant.

518 $206 VBREAK
519 $207

Die hier stehende Einsprungadresse wird verwendet, wenn die CPU eine BRK-Instruktion ausführt.

520 $208 VKEYBOARD
521 $209

Vektor für den Fall, daß eine normale Taste gedrückt wurde.

522 $20a VSERIELIN
523 $20b

Es liegt ein Zeichen im SERIN-Register von der seriellen Schnittstelle an.

524 $20c VSERREADY
525 $20d

Das SEROUT-Register ist bereit, ein weiteres Zeichen aufzunehmen.

526 $20e VSERCLOSE
527 $20f

Das parallel/seriell-Wandlungsregister ist leer und die Übertragung wird vom POKEY aus beendet.

528 $210 VTIMER1
529 $211

Signalisiert der POKEY, daß sein Zählerregister 1 auf Null gegangen ist, wird dieser Vektor angesprungen.

530 $212 VTIMER2
531 $213

Hier wird hingesprungen, wenn der POKEY im Interrupt signalisiert, daß sein Kanal 2 auf Null heruntergezählt hat.

532 $214 VTIMER4
533 $215

Erkennt die Interruptroutine beim POKEY, daß Timer 4 auf Null gezählt hat und ist der entsprechende Interrupt zugelassen, so wird dieser Vektor angesprungen.

534 $216 VIMMEDIRQ
535 $217

Hier steht die Adresse des eigentlichen Interrupthandlers, der erst die Verteilung auf die einzelnen Interruptquellen vornimmt.

536 $218 TIMCOUNT1
537 $219

Software-Timer 1. Der Wert wird bei jedem Vertical-Blank-Interrupt decrementiert. Ist er danach Null, so wird die bei (TIMER1VKT) stehende Routine ausgeführt.

538 $21a TIMCOUNT2
539 $21b

Dies ist ein ähnlicher Softwaretimer wie TIMCOUNT1, nur mit der Einschränkung, daß das Zählen dieses und der folgenden drei Zähler unterbleibt, wenn CRITICIO gesetzt ist.

Wird TIMCOUNT2 Null, so wird die bei (TIMER2VKT) stehende Routine ausgeführt.

540 $21c TIMCOUNT3
541 $21d

Für diesen Zähler gilt das gleiche wie für TIMCOUNT2, nur daß kein Vektor beim Nullwerden angesprungen wird, sondern lediglich Flag TIMER3SIG gesetzt wird.

542 $21e TIMCOUNT4
543 $21f
544 $220 TIMCOUNT5
545 $221

Es gilt das gleiche wie für TIMCOUNT3. Beim Nullwerden des entsprechenden Zählers werden analog dazu die Flags TIMER4SIG respektive TIMER5SIG gesetzt.

546 $222 VBLKIVKT
547 $223

Dieser Vertical-Blank-Immediate-Vektor wird beim Strahlrücklaufinterrupt immer als erste Unterbrechungsroutine angesprungen.

548 $224 VBLKDVKT
549 $225

Dieser Vertical-Blank-Deferred-Vektor wird von der bei VBLKIVKT stehenden Routine aus angesprungen, wenn CRITICIO gelöscht ist. Die Routine existiert im Normalfall nicht und ist frei vom Benutzer programmierbar.

550 $226 TIMER1VKT
551 $227

Ansprungvektor für die User-Routine, die ausgeführt werden soll, wenn TIMCOUNT1 auf Null gegangen ist.

552 $228 TIMER2VKT
553 $229

Ansprungvektor für die Behandlungsroutine bei Nullwerden des TIMCOUNT2.

554 $22a TIMER3SIG

Dieses Flag ist Null, wenn TIMCOUNT3 nicht Null ist, sonst $ff.

555 $22b SRTIMER

Wesentlicher, intern benutzter Timer für die Tasaturentprellung, Warten bis zum ersten Autorepeat und danach Bestimmen der Autorepeat-Rate.

556 $22c TIMER4SIG

Dieses Flag ist Null, wenn TIMCOUNT4 nicht Null ist, sonst $ff.

557 $22d IANTEMP

Temporäres Hilfsregister.

558 $22e TIMER5SIG

Dieses Flag ist Null, wenn TIMCOUNT5 nicht Null ist, sonst $ff.

559 $22f DMACNTL§

Schattenregister von DMACNTL. DMACNTL schaltet die einzelnen Arten des DMA ein.

560 $230 DLPTR§
561 $231

Schattenregister des Zeigers auf das ANTIC-Programm. Der Anfang des ANTIC-Programms muß in dieser Speicherstelle stehen, wenn mit dem Betriebssystem gearbeitet wird.

562 $232 SKCNTL§

Steuerung der seriellen Übertragung, des Ablesens der Tastatur und des Ablesens der POT-Eingänge.

563 $233 LCOUNT

Bytezähler für Loader-Routinen.

564 $234 LPENH§

Schattenregister, Horizontalposition des Lightpens.

565 $235 LPENV§

Schattenregister, Vertikalposition des Lightpens.

566 $236 VBREAKKEY
567 $237

Dies ist der Vektor für die BREAK-Tasten-Verarbeitung. Nicht verwechseln mit VBREAK, der die BRK-Instruktion der CPU behandelt!

568 $238 NEUIOINIV
569 $239

Vektor zur Initialisierungsroutine noch nicht existenter Hardware. Siehe dazu die Betriebssystembeschreibung!

570 $23a CMDDEVIV
571 $23b CMDCMD
572 $23c CMDAUX1
573 $23d CMDAUX2
574 $23e CMDAUX3
575 $23f ERRORFLAG

Interne Hilfsvariablen für die Serielldatenverarbeitung.

576 $240 DSKFLAG
577 $241 DSKSECCNT
578 $242 DSKLDADR
579 $243

Interne Hilfsvariablen für die reine Diskettenverarbeitung.

580 $244 COLDSTART

Wird dieses Flag vom Benutzerprogramm auf einen Wert ungleich Null gesetzt, so erfolgt beim Einsprung in die Warmstartroutine ein Kaltstart.

582 $246 DSKTIMCON

Kontrollregister für die Diskettenverarbeitung. Ein Wert ungleich Null besagt, daß ein Timeout aufgetreten ist.

583 $247 NEUVORHDN

Flag nur für den internen Gebrauch. Ist zu setzen, wenn externe Geräte am Parallelbus angeschlossen sind. Wird von der dazu mitgelieferten Boot-Software benutzt. Muß im Normalfall Null sein, da sonst das System abstürzen kann!

584 $248 NEUIODREQ

Auch dieses Flag wird vom Kaltstartprogramm für die Verwaltung der noch zu entwickelnden Hardware benutzt. Es enthält die Adresse des gerade angeprochenen Devices.

585 $249 NEUIOMASK
586 $24a NEULDTMP1
587 $24b

Diese Adressen werden ebenfalls nur dann sinnvoll benutzt, wenn Zusatzhardware am Parallelbus anliegt. Sie dürfen deshalb trotzdem nicht für neue Software verwendet werden.

619 $26b CHARSTPTR

Ein-Byte-Pointer für interne Monitorverarbeitung.

620 $26c FINESCROL

Temporäres Flag für die Verarbeitung des feinen Scrollings, wobei nicht gleich eine komplette Zeile verschoben wird, sondern die Zeile Punktweise nach oben gesetzt wird.

621 $26d KBDISABLE

Besitzt dieses Flag einen Wert ungleich Null, sind Eingaben von der Tastatur nicht mehr möglich.

622 $26e FINESCRFL

Wie FINESCROL für die Bildverschiebung verantwortlich.

623 $26f GTIACNTL§

Schattenregister für Steuerung des GTIA und Priorität der Spieler und Geschosse.

624	$270	PADDL0§
625	$271	PADDL1§
626	$272	PADDL2§
627	$273	PADDL3§
628	$274	PADDL4§
629	$275	PADDL5§
630	$276	PADDL6§
631	$277	PADDL7§

Schattenregister, enthalten die Werte der Paddle-Eingänge. Beim Atari 600XL und 800XL sind nur die Paddle-Eingänge 0 bis 3 benutzt, die Nummern 4 bis 7 werden von 0 bis 3 kopiert.

632	$278	JOYSTICK0
633	$279	JOYSTICK1
634	$27a	JOYSTICK2
635	$27b	JOYSTICK3

Diese Register enthalten die Stellung der Joysticks (soweit angeschlossen). Bei den neuen Atari-Modellen sind jedoch nur noch zwei Joysticks vorhanden ("0" und "1"), die beiden anderen werden kopiert.

636	$27c	PTRIG0§
637	$27d	PTRIG1§
638	$27e	PTRIG2§
639	$27f	PTRIG3§
640	$280	PTRIG4§
641	$281	PTRIG5§
642	$282	PTRIG6§
643	$283	PTRIG7§

Auslösetasten der Paddles ("0" meldet Taste gedrückt, "1" meldet Taste nicht gedrückt). Da bei den neuen Atari-Geräten nur noch 4 Paddles anzuschließen sind, werden bei diesen

nur PADDLE0 bis PADDLE3 sinnvoll verwendet, die anderen werden kopiert.

644 $284 TRIG0§
645 $285 TRIG1§
646 $286 TRIG2§
647 $287 TRIG3§

Auslösetasten der Joysticks beziehungsweise Schattenregister der GTIA-Register für die Triggereingänge ("0" meldet Taste gedrückt, "1" meldet Taste nicht gedrückt). Bei den neuen Atari-Geräten sind nur TRIG0 und TRIG1 mit Joystick-Ports verbunden, die anderen zwei Werte sind Kopien der ersten beiden.

648 $288 HIBYTELD

Zwischenregister für den LOADER.

649 $289 WRITEMODE

Flag für die Cassettenverarbeitung. Es zeigt an, ob die momentane Operation ein Schreiben ($80) oder Lesen ($00) beinhaltet. Nur für internen Gebrauch.

650 $28a CASBUFLIM
651 $28b

Wert des Pufferendes für Cassettenoperationen.

652 $28c NEUIOPTR
653 $28d

Pointer auf die Startadresse der I/O-Verarbeitung der neuen Hardwarezusätze am parallelen Bus.

654 $28e NEWADRLOD
655 $28f

Ladeadresse für I/O mit der neuen Zusatzhardware.

656 $290 TEXTROW
657 $291 TEXTCOL
658 $292
659 $293 TEXTINDEX
660 $294 TEXTMSC
661 $295
662 $296 TEXTOLD
663 $297
666 $29a TEXTGRAD

Variablen zur Verarbeitung von Text/Grafik-gemischten Bildschirminhalten.

667 $29c CRETRY

Dies ist die Maximalzahl der Wiederholungen des Versuches, ein Kommando auf einem sich zurückmeldenen Gerät abzusetzen. Der Wert steht auf 1, so daß eine Wiederholung stattfindet. Mehr ist hierbei auch nicht sinnvoll, denn es ist davon auszugehen, daß ein Gerät, das sich ordentlich zurückmeldet, im Normalfall auch in der Lage ist, ein richtiges Kommando auszuführen. Gelingt dies nicht, sind das Kommando oder aber Voraussetzungen für die richtige Ausführung nicht in Ordnung.

668 $29e SUBTEMP
669 $29f HOLD2

Hilfsvariablen für interne Verwendung.

670 $2a0 DISPLYMSK
671 $2a1 TEMPLBT
672 $2a2 ESCAPEFLAG

Weitere Variablen zur reinen Textverarbeitung mit Screen oder Editor.

673 $2a3 TABMAP
bis
689 $2b1

Tabelle, in der jedes einzelne Bit eine Cursorposition innerhalb einer logischen Zeile auf dem Bildschirm darstellt. Ist ein Bit gesetzt, so gilt diese Position als Tabulator-Haltestelle.

690 $2b2 LOGICMASK
bis
693 $2b5

Tabelle für die Zusammenhänge von logischer und physikalischer Zeile auf dem momentan angezeigten Bildschirm. Wird intern benötigt, um beim Löschen einer Zeile immer eine komplette logische Zeile löschen zu können.

694 $2b6 XORKEYMSK

Diese Variable wird verwendet, um die Funktionalität von Tastengruppen der Tastatur abändern zu können.

Jedes gesetzte Bit in diesem Byte gibt an, daß das entsprechende, vom POKEY gelieferte Datenbit aus dem Tastaturkode invertiert werden soll. So können ganze Tastaturreihen in andere umgewandelt werden. Es muß im Einzelnen herausgefunden werden, in wieweit diese Funktion gewinnbringend eingesetzt werden kann. Es ist jedoch zu beachten, daß diese Invertierung der jeweiligen Bits mit dem Tastaturkode geschieht, also bevor der gelesene Wert in einen ATASCII-Wert umgewandelt wird.

701 $2bd DRETRY

Hier steht die Anzahl, mit der der Versuch wiederholt wird, ein spezifiziertes Gerät anzusprechen. Der Wert ist im Normalbetrieb auf 13 Wiederholungen eingestellt.

702 $2be SHIFTLOCK

Dieses Label gibt Auskunft darüber, ob irgendwelche Sonderbedingungen bei der Eingabe zu beachten sind:

$00: Kleinbuchstaben aktiv (Schreibmaschinenmodus)

$40: Nur Großbuchstaben (Normalmodus)

$80: CONTROL-Taste gedrückt.

703 $2bf NUMNXTLIN

Dieses Register beinhaltet die Anzahl der auf dem Bildschirm verarbeiteten Textzeilen.

Das Register kann die Werte 24, 4 oder 0 annehmen, andere werden vom Betriebssystem ignoriert.

704	$2c0	COLPM0§
705	$2c1	COLPM1§
706	$2c2	COLPM2§
707	$2c3	COLPM3§

Schattenregister der Farbregister für die Spieler beziehungsweise Geschosse.

708	$2c4	COLPF0§
709	$2c5	COLPF1§
710	$2c6	COLPF2§
711	$2c7	COLPF3§
712	$2c8	COLBAK§

Schattenregister der Farbregister für die Spielfeldfarben beziehungsweise des Hintergrundes.

713	$2c9	RUNADRLOD
714	$2ca	
715	$2cb	HIUSEDLOD
716	$2cc	
717	$2cd	LODZHIUSE
718	$2ce	
719	$2cf	LODGBYTEA
720	$2d0	
721	$2d1	LODADRESS
722	$2d2	
723	$2d3	LODZLOADA
724	$2d4	

Diese Variablen werden für interne Verwaltungsaufgaben beim Programmladen der später einmal am hinteren parallelen Bus anzuschließenden Hardware verwendet.

725 $2d5 DSKSECLEN
726 $2d6

Dieser Wert kann normalerweise $0080 oder $0100 sein, je nach Diskettenformat.

727 $2d7 ACMISR
728 $2d8

Findet interne Verwendung.

729 $2d9 KEYRPDELY

Dieser Wert bestimmt die Zeit zwischen dem Drücken einer Taste und dem ersten Zeichen des Autorepeat. Standardwert ist 40.

730 $2da KEYREP

Im Gegensatz zu KEYRPDELY bestimmt dieser Wert bei einsetzendem Autorepeat die Wiederholrate nach dem ersten Auto-Zeichen. Standardwert ist 5.

731 $2db CLICKDISA

Wenn ungleich Null, ist der Tastaturklick abgeschaltet.

732 $2dc HELPFLAG

Dieses Register ist beim 600XL/800XL 'Entwicklungsschrott', da es angibt, ob die in diesen Geräten nicht eingebaute HELP-Taste gedrückt ist. Darf trotzdem nicht verwendet werden, da unter Umständen Schreibzugriffe darauf programmiert sind.

733 $2dd DMASAVE

Statusinformation des Direkten Speicherzugriffs. Hat für normalen Benutzer keinen Nutzen, darf aber nicht anderweitig benutzt werden.

734 $2de PRTBUFPTR

Gibt Bytenummer innerhalb des Printerpuffers an.

735 $2df PRTBUFSIZ

Ist maximale Printerpuffergröße in Byte.

740 $2e4 RAMSIZE

Anzahl der zur Verfügung stehenden RAM-Pages des Systems.

741 $2e5 MEMTOP
742 $2e6

Dies ist ein Zeiger auf die Obergrenze des vom Benutzer verwendbaren Speicherraumes. Darüber liegt im Normalfall der Bildschirm.

743 $2e7 MEMLO
744 $2e8

Dies ist der Zeiger auf die Untergrenze des vom Benutzer frei zu verfügenden Speicherraumes. Er liegt normalerweise direkt über den Betriebssystemvariablen, respektive dem DOS, eventuell noch mit DUP.

745 $2e9 HNDLRLOAD
746 $2ea DEVICSTAT
747 $2eb
748 $2ec CHAINTMP
749 $2ed

Hilfsvariablen für den internen Gebrauch bei der Verwaltung linearer Listen und darüber laufender Ladevorgänge mit noch nicht existenter Hardware.

750 $2ee CASSPEED
751 $2ef

Zeitwerte für die Senderate über Cassette.

752 $2f0 CURSORINH

Auf einem Wert ungleich Null, verhindert diese Variable die Anzeige des Cursors.

753 $2f1 KEYDELAY

Dieser Wert, der standardmäßig auf 3 eingestellt ist, bestimmt die Zeit zwischen dem Loslassen einer Taste und dem Drücken einer neuen Taste, so daß deren Kode übernommen wird. Bei extrem schnellen 'Tippern' (und 'Tipperinnen') sollte dieser Wert verkleiner werden. Das verhindert Ärger mit fehlenden Zeichen beim Programmieren.

755 $2f3 CHARCNTL$

Schattenregister, steuert das Aussehen der Zeichen in mehreren Schrift-Modi des ANTIC.

756 $2f4 CHARBASE§

Schattenregister, enthält Basisadresse des Zeichengenerators (oberes Byte)

757 $2f5 NEWGRROW
758 $2f6 NEWGRCOL
759 $2f7

Hier werden die Werte für einen DRAWTO-Befehl vor dessen Aufruf festgelegt.

760 $2f8 ROWINC
761 $2f9 COLINC
762 $2fa

Diese Variablen werden zum Berechnen der Bahnen eines DRAWTO benutzt.

763 $2fb ATASCICHR

Letztes eingegebenes Zeichen im ATASCII-Kode. Fast ausschließlich für interne Zwecke benutzt.

764 $2fc KBCODE§

Schattenregister, enthält den Tastaturcode, das heißt, welche Taste gedrückt wurde.

765 $2fd FILEDAT

Für die interne Benutzung des Grafiksystems reserviert.

766 $2fe DISPLYFLG

Ist dieses Flag ungleich Null, so wird das folgende Zeichen, wenn es ein Kontrollzeichen ist, nicht ausgeführt, sondern der entsprechende ATASCII-Kode ausgegeben.

767 $2ff STARTSTOP

Ist dieses Byte gleich $ff, so storniert die Bildschirmausgabe, bei $00 geht sie weiter. das Flag wird durch Drücken von CONTROL & '1' verändert. Wichtig ist, daß bei der momentanen Betriebssystemimplementierung das System 'steht', wenn es auf das Nullwerden von STARTSTOP wartet. Nur wesentliche Interrupts werden durchgelassen, die eigentliche Rechenarbeit wird jedoch ebenfalls mit dem Output angehalten.

Im Folgenden werden die wesentlichen, vom Disk-Handler benutzten Variablen beschrieben. Sie sind größtenteils bei Diskettenverarbeitung selbst zu belegen, es sei denn, die Programmierung erfolgt in einer höheren Sprache, wie zum Beispiel BASIC oder PASCAL.

768 $300 DSKDEVICE

Allgemeiner Erkennungskode für die Disketten- ($30) beziehungsweise Cassettenstation ($60).

769 $301 DSKUNIT

Bei Diskettenverarbeitung die Nummer der anzusprechenden Diskettenstation.

Erlaubt sind hier Werte im Bereich von $01 bis $09, wobei bei der heutigen Hardware nur $01 bis $04 sinnvoll, weil kaufbar sind.

770 $302 DSKCMD

Auszuführendes Kommando. Siehe dazu Zeropage-IOCBs ab IOCBCHIDZ.

771 $303 DSKSTATUS

Zurückgemeldeter Status der Operation. Siehe dazu ebenfalls IOCBSTATZ.

772 $304 DSKBUFFER
773 $305

Anfangsadresse des Datenpuffers.

774 $306 DSKTIMOUT
775 $307

Anzahl der Sekunden bis zur Timeout-Fehlermeldung.

776 $308 DSKBYTCNT
777 $309

Anzahl der im Puffer ab DSKBUFFER zur Verfügung stehenden Bytes.

778 $30a DSKAUX1
779 $30b DSKAUX2

Hilfsinformationen für die Diskettenverarbeitung. Bei OPEN-Befehlen stehen hier besondere Merkmale (z.B. Nur-Lese-Zugriff).

Bei Schreib- beziehungsweise Leseoperationen steht hier in DSKAUX1 und -2 die Blocknummer von $01 bis MAXBLOCK.

780 $30c INTERVTI1
781 $30d

Intervalltimer 1, bildet mit INTERVTI2 eine Einheit.

782 $30e OPTIONJMP

Diese Adresse ist offiziell von Atari als belegt gekennzeichnet, ist jedoch nirgendwo im Betriebssystem verwendet. Sie soll für spätere Hardware irgendeine Vorentscheidung treffen. Es empfiehlt sich im Hinblick auf spätere Softwarekompatibilität, diese Variable nicht zu verwenden.

783 $30f CASFLAG

Ist dieses Flag nicht Null, handelt es sich um eine Cassettenoperation. Nur für internen Gebrauch.

784 $310 INTERVTI2
785 $311

INTERVTI1 und -2 werden benutzt, um die Lesegeschwindigkeit bei Cassettenoperationen zu bestimmen.

786 $312 CHAINTP1
787 $313

Hilfspointer für lineare-Listenverwaltung.

788 $314 PTIMOUT

Timeoutwert für Druckbefehle.

791 $317 TIMEFLAG

Timeoutwert für die Lesegeschwindigkeitsbestimmung.

792 $318 STACKSAVE

Hilfsregister für das Retten des Stackpointers bei SIO-Operationen.

793 $319 TEMPSTAT

Register für kurzzeitiges Zwischenspeichern von SIO-Statusinformationen.

794 $31a bis 831 $33f HATABS

Tabelle für die Zuordnung von Geräten zu deren Handlern. Jeder Tabelleneintrag besteht aus drei Byte: Dem Namen (1) und der Handlertabellenstartadresse (2). Es sind die Geräte Cassette, Editor, Screen, Drucker und Keyboard vorgegeben. Ein Eintrag ist dann leer, wenn sein Name Null ist.

832 $340 bis 847 $34f IOCB0

Ab dieser Adresse stehen 8 IOCB-Einträge zur Verfügung, von denen Eintrag 0 bereits vom System für die Editorverarbeitung verwendet

wird. Die einzelnen Einträge bitte aus der Zeropage-IOCB-Beschreibung ersehen, die ab IOCBCHIDZ steht. Bei Verwendung eines dieser IOCBs wird davor der Inhalt aus diesen Registern in die Zeropage kopiert, um sie danach wieder zurückzutransportieren.

848 $350 IOCB1
bis
863 $35f

864 $360 IOCB2
bis
879 $36f

880 $370 IOCB3
bis
895 $37f

896 $380 IOCB4
bis
911 $38f

912 $390 IOCB5
bis
927 $39f

928 $3a0 IOCB6
bis
943 $3af

944 $3b0 IOCB7
bis
959 $3bf

960 $3c0 bis 999 $3e7 — PRINTBUF

Hier liegt der Printerpuffer für die Ausgabe an den Drucker über die serielle Schnittstelle.

1000 $3e8 — SUPERFLAG

Hilfsinformation für Editor.

1001 $3e9 — STARTTST

Dieses Flag liefert einen Wert ungleich Null, wenn beim Kaltstart die START-Taste gedrückt war.

1002 $3ea — CASSTART

Flag, ob gerade von Cassette gebootet wird und ob die dort gelesene Init-Adresse angesprungen werden soll.

1003 $3eb
1004 $3ec — CARTCKSUM

Checksumme des Cartridge mit dem Anfang des ROMs bei $C000.

1005 $3ed bis 1015 $3f7 — ACMVAR

Hilfsvariablen für interne Zwecke.

1016 $3f8 X64KBFLAG

Flag, ob die obersten 16KByte RAM oder ROM sind.

1017 $3f9 MINTLK

Hilfsregister, nicht benutzen.

1018 $3fa TRIG3§

Dies ist eine Kopie des Registers TRIG3, das normalerweise nur durch einen Eingang beeinflußt wird. Wird ein ROM-Modul eingesteckt ($01) oder entfernt ($00), so ändert sich der Wert von TRIG3. Wird zur Feststellung benutzt, ob es sich um einen Kalt- oder Warmstart handelt.

1019 $3fb
1020 $3fc CHAINLINK

Pointer zur Verarbeitung linearer Listen, wenn sie eingesetzt werden; also nur bei Verwendung spezieller, noch nicht auf dem Markt befindlicher Hardware.

1021 $3fd
bis
1151 $47f CASBUFFER

Pufferbereich für die Cassettendatenübertragung, wobei die drei ersten Byte nur für die Synchronisation und Feststellung der Datentypen und -menge gelten. Der eigentliche Datenpuffer beginnt ab Adresse 1024 ($400) und geht bis zum Ende von CASBUFFER.

An dieser Stelle sind die Variablen des Betriebssystems beendet. Sicherlich werden noch diverse andere Speicherstellen in der Zeropage und im Bereich über $0480 benutzt. Da diese Benutzung jedoch auf BASIC, DOS oder andere Software beschränkt ist, wird an dieser Stelle nicht weiter darauf eingegangen.

Im Weiteren folgen die Hardwareadressen von GTIA, POKEY, PIA und ANTIC:

GTIA - Adressbereich

53248 $d000 HPOSP0 (W)
53249 $d001 HPOSP1 (W)
53250 $d002 HPOSP2 (W)
53251 $d003 HPOSP3 (W)

53252 $d004 HPOSM0 (W)
53253 $d005 HPOSM1 (W)
53254 $d006 HPOSM2 (W)
53255 $d007 HPOSM3 (W)

Horizontalpositionen der Spieler (HPOSPn) beziehungsweise der Geschosse (HPOSMn).

53248 $d000 KOLM0PF (R)
53249 $d001 KOLM1PF (R)
53250 $d002 KOLM2PF (R)
53251 $d003 KOLM3PF (R)

Kollisionsregister für Zusammenstöße zwischen den Geschossen und dem Spielfeld.

53252 $d004 KOLP0PF (R)
53253 $d005 KOLP1PF (R)
53254 $d006 KOLP2PF (R)
53255 $d007 KOLP3PF (R)

Kollisionsregister für Zusammenstöße zwischen den Spielern und dem Spielfeld.

53256 $d008 SIZEP0 (W)
53257 $d009 SIZEP1 (W)
53258 $d00a SIZEP2 (W)
53259 $d00b SIZEP3 (W)

53260 $d00c SIZEM (W)

Größen der Spieler (SIZEPn) und der Geschosse (SIZEM).

53256 $d008 KOLM0PL (R)
53257 $d009 KOLM1PL (R)
53258 $d00a KOLM2PL (R)
53259 $d00b KOLM3PL (R)

Kollisionsregister für Zusammenstöße zwischen Geschossen und Spielern.

53261 $d00d GRAFP0 (W)
53262 $d00e GRAFP1 (W)
53263 $d00f GRAFP2 (W)
53264 $d010 GRAFP3 (W)

53265 $d011 GRAFM (W)

Grafikregister der Spieler (GRAFPn) beziehungsweise der Geschosse (GRAFM).

53260 $d00c KOLP0PL (R)
53261 $d00d KOLP1PL (R)
53262 $d00e KOLP2PL (R)
53263 $d00f KOLP3PL (R)

Kollisionsregister für Zusammenstöße zwischen Spielern.

53264 $d010 TRIG0 (R)
53265 $d011 TRIG1 (R)
53266 $d012 TRIG2 (R)
53267 $d013 TRIG3 (R)

Register zum Abfragen der Triggereingänge des GTIA (Schattenregister dazu ab 644 beziehungsweise $284).

53266 $d012 COLPM0 (W)
53267 $d013 COLPM1 (W)
53268 $d014 COLPM2 (W)
53269 $d015 COLPM3 (W)

Farben der Spieler beziehungsweise der Geschosse (Schattenregister dazu ab 704 beziehungsweise $2c0).

53270 $d016 COLPF0 (W)
53271 $d017 COLPF1 (W)
53272 $d018 COLPF2 (W)
53273 $d019 COLPF3 (W)

53274 $d01a COLBAK (W)

Farben des Spielfeldes (COLPFn) und des Hintergrundes (COLBAK), Schattenregister dazu ab 708 beziehungsweise $2c4.

53275 $d01b GTIACNTL (W)

Steuert den GTIA, bestimmt Prioritätsfolge zwischen Bildelementen und steuert Darstellung der Spieler und Geschosse.

53276 $d01c VDELAY (W)

Ermöglicht es, Spieler und Geschosse um einzelne Bildzeilen zu verschieben, wenn zweizeilige Auflösung der Player-Missile-Grafik gewählt wurde.

53277 $d01d PMCNTL (W)

Schaltet Spieler und Geschosse ein; ermöglicht es, den Status der Triggereingänge festzuhalten.

53278 $d01e HITCLR (W)

Wenn in dieses Register irgendein Wert geschrieben wird (wenn diese Adresse im Schreibzugriff angesprochen wird), werden alle Kollisionsregister gelöscht.

53279 $d01f CONSOL (W/R)

Dieses Register wird verwendet, um den Status der Konsolentaster ("START", "SELECT" und "OPTION") abzufragen.

NEUPORT-Adressbereich

53759 $d1ff NEUPORT

Dieses Register dient der I/O-Erweiterung. Siehe Betriebssystembeschreibung!

POKEY-Adressbereich

53760 $d200 AUDIFREQ1 (W)
53761 $d201 AUDIFREQ2 (W)
53762 $d202 AUDIFREQ3 (W)
53763 $d203 AUDIFREQ4 (W)

Diese Register bestimmen die Frequenz der Tongeneratoren.

53764 $d204 AUDICNTL1 (W)
53765 $d205 AUDICNTL2 (W)
53766 $d206 AUDICNTL3 (W)
53767 $d207 AUDICNTL4 (W)

Diese Register steuern die einzelnen Tongeneratoren. Mit diesen Registern bestimmt man jeweils für einen Tongenerator die Lautstärke, Verzerrungen beziehungsweise den VOLUME_ONLY-Modus.

53760 $d200 POT0 (R)
53761 $d201 POT1 (R)
53762 $d202 POT2 (R)
53763 $d203 POT3 (R)
53764 $d204 POT4 (R)
53765 $d205 POT5 (R)
53766 $d206 POT6 (R)
53767 $d207 POT7 (R)

Diese Register enthalten den Wert der POT-Eingänge des POKEY. Bei den neuen Atari-Geräten werden nur noch die POT-Eingänge 0 bis 3 verwendet, da auch nur noch zwei Joystick-Ports zur Verfügung stehen. Die vier verbleibenden Register beinhalten Kopien der ersten vier.

53768 $d208 AUDICOM (W)

Dieses Register steuert die Tonerzeugung im Atari. Mit diesem Register werden vor allem die Grundfrequenzen für die vier Tonkanäle eingestellt und Hochtonfilter ein- beziehungsweise ausgeschaltet.

53768 $d208 POTSTAT (R)

Mit diesem Register läßt sich bestimmen, ob für einen bestimmten POT-Eingang die Analog-Digital-Wandlung bereits abgeschlossen ist. Wenn ein Bit von POTSTAT '0' ist, bedeutet dies, daß der Wert des dazugehörigen POT-Eingang gültig ist. Die Bits sind folgendermaßen zugeordnet:

Bit 0 : POT-Eingang 0
Bit 1 : POT-Eingang 1
Bit 2 : POT-Eingang 2
. .
. .
Bit 7 : POT-Eingang 7

53769 $d209 STIMER (W)

Durch Ansprechen dieser Adresse in einem Schreibzugriff werden die Audio-Frequenz-Teiler auf ihre "AUDIFREQ"-Werte zurückgesetzt.

53769 $d209 KBCODE (R)

Dieses Register enthält den Tastaturcode, das heißt, der Benutzer kann hier abfragen, welche Taste gedrückt ist (Schattenregister bei 764 beziehungsweise $2fc).

53770 $d20a SKSTATRES (W)

Durch Eintragen irgendeines Wertes in dieses Register werden die Bits 7,6 und 5 von SKSTAT gelöscht.

53770 $d20a RANDOM (R)

Dieses Register enthält eine "quasi"-zufällige Zahl (8 Bit), die aus einem 17- beziehungsweise 9-Bit-Polynom-Zähler stammt.

53771 $d20b POTGO (W)

Durch Eintragen irgendeines Wertes in dieses Register wird der Ablesevorgang der POT-Eingänge gestartet.

53773 $d20d SEROUT (W)

Ausgaberegister der seriellen Schnittstelle.

53773 $d20d SERIN (R)

Eingaberegister der seriellen Schnittstelle.

53774 $d20e IRQEN (W)

Die Bits in diesem Register schalten die einzelnen Arten zur Auslösung eines IRQ ein beziehungsweise aus.

53774 $d20e IRQSTAT (R)

Nach dem Auftreten eines IRQ kann man durch Abfragen dieses Registers die Herkunft der Unterbrechungsanforderung bestimmen.

53775 $d20f SKCNTL (W)

Durch dieses Register lassen sich die einzelnen Betriebsmodi der seriellen Schnittstelle, der POT-Wert-Wandlung und der Tastatur-Abfrage wählen (Schattenregister bei 562 beziehungsweise $262).

53775 $d20f SKSTAT (R)

Dieses Register enthält Angaben zum Status der seriellen Schnittstelle sowie der Tastatur. Die Bit 7,6 und 5 werden mit SKSTATRES (53770 beziehungsweise $d20a) zurückgesetzt.

PIA-Adressbereich

54016 $d300 PORTA (R/W)

Übertragungsregister "A" der PIA.

54017 $d301 PORTB (R/W)

Übertragungsregister "B" der PIA.

54018 $d302 PORTACNTL

Steuer- und Statusregister für den PIA-Port "A".

54019 $d303 PORTBCNTL

Steuer- und Statusregister für den PIA-Port "B".

ANTIC-Adressbereich

54272 $d400 DMACNTL (W)

Mit diesem Register kann man den DMA des ANTIC steuern (Schattenregister bei 559 beziehungsweise $22f).

54273 $d401 CHARCNTL (W)

Mit diesem Register kann man das Aussehen der Zeichen in mehreren ANTIC-Modi beeinflußen (Schattenregister bei 755 beziehungsweise $2f3).

54274 $d402 DLPTRL
54275 $d403 DLPTRH

In diesen beiden Registern steht die 16-Bit-Adresse des ANTIC-Programms (Schattenregister bei 560/561 beziehungsweise $230/$231). Dies ist also der Zeiger auf das ANTIC-Programm.

54276 $d404 HSCROL (W)

Mit diesem Register bestimmt man, um wieviele Farbtakte ein bestimmter Teil des Bildes nach rechts verschoben werden soll. Zu verschiebende Zeilen müssen im ANTIC-Programm gekennzeichnet werden.

54277 $d405 VSCROL (W)

Mit diesem Register bestimmt man, um wieviele Bildzeilen ein bestimmter Teil des Bildes nach oben geschoben werden soll. Die zu verschiebenden Zeilen müssen im ANTIC-Programm gekennzeichnet werden.

54279 $d407 PMBASE (W)

In dieses Register bringt man die oberen acht Bits der Basisadresse des Speicherfeldes für die Player-Missile-Grafik.

54281 $d409 CHARBASE (W)

In dieses Register müssen die oberen 8 Bits der Basisadresse des Zeichengenerators gebracht werden (Schattenregister bei 756 beziehungsweise $2f4).

54282 $d40a WAITHSYNC (W)

Durch Ansprechen dieses Registers wird die CPU bis zum Beginn der nächsten Horizontalsynchronisation angehalten.

54283 $d40b VCOUNT (R)

Dieses Register enthält die Nummer der Bildzeile, die gerade erzeugt wird, geteilt durch zwei.

54284 $d40c LPENH (R)

Dieses Register enthält die Horizontalposition des Lightpens (Schattenregister bei 564 beziehungsweise $234).

54285 $d40d LPENV (R)

Dieses Register enthält die Vertikalposition (Nummer der Bildzeile geteilt durch zwei) des Lightpens (Schattenregister bei 565 beziehungsweise $235).

54286 $d40e NMIEN (W)

Mit diesem Register werden die Unterbrechungen bei der Vertikalsynchronisation und bei ANTIC-Programm-Unterbrechungen ein- beziehungsweise ausgeschaltet.

54287 $d40f NMIST (R)

Nach dem Auftreten eines NMI kann man durch Abfragen dieses Registers die Herkunft der Unterbrechungsanforderung bestimmen.

54287 $d40f NMIRES (W)

Durch Eintragen irgendeines Wertes in dieses Register wird NMIST wieder gelöscht.

CPU-Hardwarevektoren

65530 $fffa NMIVKT
65531 $fffb

Nach einem NMI springt die CPU zu der Adresse, die sich in NMIVKT befindet (das niederwertige Byte steht an erster Stelle).

65532 $fffc RESETVKT
65533 $fffd

Nach einem RESET springt die CPU zu der Adresse, die sich in RESETVKT befindet (das niederwertige Byte steht an erster Stelle).

65534 $fffe IRQVKT
65535 $ffff

Nach einem IRQ springt die CPU zu der Speicherstelle, die sich in IRQVKT befindet (das niederwertige Byte steht an erster Stelle).

D A S   R E G I S T E R   D E R   L A B E L S

ABUFPTR1	$001D	BITGET	$F75D
ABUFPTR2	$001E	BITMASK	$006E
ABUFPTR3	$001F	BITMASKE	$CA2F
ACMISR	$02D7	BITPUT	$F73C
ACMVAR	$03ED	BITROL	$F732
ACTCHRNUM	$0063	BLOAD	$C637
ADD28E	$C87B	BLOCK1	$C5C9
ADD28EGET	$C8C3	BOOT	$C599
ADD28EPUT	$C8E0	BOTTOMLIN	$F55F
ADD28EWRD	$C8A0	BRKEVENT	$C092
ADDRESS	$0064	BUFENDPTR	$0034
ADJUST	$ED2E	BUFFERADR	$0015
ADJUSTTAB	$EE11	BUFFULL	$0038
AKTBUFLEN	$006B	BUFSTR	$006C
ATASCICHR	$02FB	CALLTAB	$E48F
ATRAKTMSK	$004E	CALLVKT	$E900
ATTRACT	$004D	CALLVKTC1	$E894
AUDICNTL1	$D204	CARTCKSUM	$03EB
AUDICNTL2	$D205	CARTGO	$C47F
AUDICNTL3	$D206	CASBOOT	$C67C
AUDICNTL4	$D207	CASBUFLIM	$028A
AUDICOM	$D208	CASBUFPTR	$003D
AUDIFREQ1	$D200	CASCLOSE	$FDCF
AUDIFREQ2	$D201	CASDATA	$0400
AUDIFREQ3	$D202	CASENTER	$EB9D
AUDIFREQ4	$D203	CASEOF	$003F
BASMEMTOP	$000E	CASFLAG	$030F
BEEPCOUNT	$0040	CASINIT	$C6AE
BEEPWAIT	$FDFC	CASINITV	$0002
BEGINREAD	$ED3D	CASMOTOFF	$FD05
BELL	$F556	CASOPEN	$FCE6
BITCLR	$F74A	CASPUTBYT	$FDB4
BITCON	$F723	CASRDBYTE	$FD7A

DCBXINIT	$E833	DSKFLAG	$0240
DECBFP	$E6C8	DSKLDADR	$0242
DECBUFL	$E6BB	DSKRDERR	$C64C
DECTIMER	$C263	DSKSECCNT	$0241
DELCHAR	$F4D5	DSKSECLEN	$02D5
DELLINE	$F520	DSKSTAT	$0030
DELTIA	$F918	DSKSTATUS	$0303
DERRMSG	$C44B	DSKTIMOUT	$0246
DEVICSRCH	$E712	DSKTIMOUT	$0306
DEVICSTAT	$02EA	DSKUNIT	$0301
DEVS2PL3	$E6FF	EDITORVKT	$E400
DISKFORM	$0018	EGETCH	$F24A
DISKINIT	$C6B1	ENDPOINTR	$0074
DISKINTERF	$C6C1	EOUTCH	$F2B0
DISKUTIL	$001A	ERANGE	$F6BC
DISPLYFLG	$02FE	ERRORFLAG	$023F
DISPLYMSK	$02A0	ESCAPE	$F3E0
DLIVKT	$0200	ESCAPEFLG	$02A2
DLPTR§	$0230	EXITVBL	$C298
DLPTRH	$D403	EXTEND	$F7C2
DLPTRL	$D402	FILEDAT	$02FD
DMACNTL	$D400	FILEMNGMT	$0043
DMACNTL§	$022F	FINESCRFL	$026E
DMASAVE	$02DD	FINESCROL	$026C
DOBUFC	$F8B1	FKDEF	$FB11
DOSAKTIV	$0009	FKTDEFPTR	$0060
DOSCROLL	$F7F7	FREI0	$CB73
DOSINITC	$C649	FREI1	$EF65
DOSINITV	$000C	FREI2	$FCD6
DOSVKT	$000A	GAPTYPE	$003E
DOSVKTC	$C434	GETBLOCK	$C667
DRAWTO	$F9AF	GETBYTEAC	$C7DD
DRETRY	$02BD	GETCH	$F180
DSKAUX1	$030A	GETCHECKS	$FFA4
DSKAUX2	$030B	GETLOWEST	$C9BD
DSKBUFFER	$0304	GETPLT	$F18F
DSKBUFPTR	$0032	GETRAMHI	$C4B7
DSKBYTCNT	$0308	GOHANDLER	$E6EA
DSKCHKSUM	$0031	GOMEMTEST	$C3BD
DSKCMD	$0302	GRAFM	$D011
DSKDEVICE	$0300	GRAFP0	$D00D

Label	Adresse
GRAFP1	$D00E
GRAFP2	$D00F
GRAFP3	$D010
GRAPHEMUL	$0057
GTIACNTL	$D01B
GTIACNTL§	$026F
HANDLERTB	$E440
HATABS	$031A
HELPFLAG	$02DC
HIBYTELD	$0288
HILFSWORT	$0000
HITCLR	$D01E
HIUSEDLOD	$02CB
HNDLRLOAD	$02E9
HOLD2	$029F
HPOSM0	$D004
HPOSM1	$D005
HPOSM2	$D006
HPOSM3	$D007
HPOSP0	$D000
HPOSP1	$D001
HPOSP2	$D002
HPOSP3	$D003
HSCROL	$D404
IANTEMP	$022D
INATAC	$F76A
INCBFP	$E6D1
INCLOAD	$E816
INCRSB	$F60A
INDSETVKT	$E912
INIT	$FCDB
INIT200	$C459
INIT31A	$C34C
INITCARTC	$C437
INITLOAD	$E7DE
INITPOTS	$C239
INITSOME	$EF8E
INSCHAR	$F49F
INSLINE	$F50C
INTERCHAR	$CC00
INTTAB	$ECA9
INTVKT	$FFFE
IOCB0	$0340
IOCB1	$0350
IOCB2	$0360
IOCB3	$0370
IOCB4	$0380
IOCB5	$0390
IOCB6	$03A0
IOCB7	$03B0
IOCBAUX1Z	$002A
IOCBAUX2Z	$002B
IOCBAUX3Z	$002C
IOCBAUX4Z	$002D
IOCBBUFAZ	$0024
IOCBBUFLZ	$0028
IOCBCHARZ	$002F
IOCBCHIDZ	$0020
IOCBCMD	$0017
IOCBCMDZ	$0022
IOCBDSKNZ	$0021
IOCBNUMZ	$002E
IOCBPUTBZ	$0026
IOCBSTATZ	$0023
IOPORTINI	$C4E8
IOSOUNDEN	$0041
IRQEN	$D20E
IRQEN§	$0010
IRQST§	$0011
IRQSTAT	$D20E
ISRODN	$EAAD
ISRSIR	$EB2C
ISRXD	$EAEC
JMPCASOFF	$EF6B
JMPF21E	$F715
JMPF983	$FCD8
JMPIRQVKT	$C02C
JMPSIOINT	$E959
JMPTAB	$E450
JMPTIMER1	$C25D
JMPTIMER2	$C260
JOYSTICK0§	$0278

JOYSTICK1§	$0279
JOYSTICK2§	$027A
JOYSTICK3§	$027B
JSRIND	$F2AD
KBCODE	$D209
KBCODE§	$02FC
KBDISABLE	$026D
KBGETCHAR	$F302
KBVKT	$E420
KEYCLICK	$F983
KEYDEF	$FB51
KEYDEFPTR	$0079
KEYDELAY	$02F1
KEYREP	$02DA
KEYRPDELY	$02D9
KOLM0PF	$D000
KOLM0PL	$D008
KOLM1PF	$D001
KOLM1PL	$D009
KOLM2PF	$D002
KOLM2PL	$D00A
KOLM3PF	$D003
KOLM3PL	$D00B
KOLP0PF	$D004
KOLP0PL	$D00C
KOLP1PF	$D005
KOLP1PL	$D00D
KOLP2PF	$D006
KOLP2PL	$D00E
KOLP3PF	$D007
KOLP3PL	$D00F
LCOUNT	$0233
LFTMARGIN	$0052
LINEINSERT	$F78E
LINK	$E898
LINKSOMETH	$E739
LOADER	$C753
LOADERTMP	$0036
LOADPTR	$EB87
LODADRESS	$02D1
LODGBYTEA	$02CF
LODZHIUSE	$02CD
LODZLOADA	$02D3
LOGGET	$F758
LOGICMASK	$02B2
LOWMEM	$0080
LPENH	$D40C
LPENH§	$0234
LPENV	$D40d
LPENV§	$0235
LPTVKT	$E430
MASKTAB	$C0CF
MATHMUELL	$D805
MEMLO	$02E7
MEMTOP	$02E5
MINTLK	$03F9
MLTTMP	$0066
MONSTATUS	$004C
MONTEMP1	$0051
NEUDEVC1	$C99F
NEUDEVC3	$C9A4
NEUDEVC5	$C9A9
NEUDEVC7	$C9AE
NEUDEVC9	$C9B3
NEUDEVCB	$C9B8
NEUINIT	$C91A
NEUINITC	$E49B
NEUIODREQ	$0248
NEUIOINIV	$0238
NEUIOMASK	$0249
NEUIOPTR	$028C
NEUIOREQ	$C97C
NEUPORT	$D1FF
NEUPORTERR	$C9D8
NEUPORTST	$CA11
NEUVECTOR	$C8F2
NEUVRORHDN	$0247
NEWADRLOD	$028E
NEWCART	$C4D7
NEWDEVICE	$EEBC
NEWGRCOL	$02F6
NEWGRROW	$02F5

NEWLDTMP1	$024A
NMIEN	$D40E
NMIENABLE	$C00C
NMIRES	$D40F
NMIST	$D40F
NMIVKT	$FFFA
NOCHKSUM	$003C
NTSCPAL§	$0062
NUMTXTLIN	$02BF
OFFCURSOR	$F718
OLDGRADR	$005E
OLDGRCHR	$005D
OLDGRCOL	$005B
OLDGRROW	$005A
OPTIONJMP	$030E
OUTCH	$F1A4
OUTPLT	$F1CA
PADDLE0§	$0270
PADDLE1§	$0271
PADDLE2§	$0272
PADDLE3§	$0273
PADDLE4§	$0274
PADDLE5§	$0275
PADDLE6§	$0276
PADDLE7§	$0277
PADTRIGS	$C23C
PHACRS	$F94C
PHCLOSE	$FF02
PHOPEN	$FEC2
PHPUT	$FF3F
PHSTAT	$FEA3
PHWRITE	$FECB
PLACRS	$F957
PLOTCOLAC	$0072
PLOTROWAC	$0070
PMBASE	$D407
PMCNTL	$D01D
PNMI	$C018
POKTAB	$EDF9
PORTA	$D300
PORTACNTL	$D302

PORTB	$D301
PORTBCNTL	$D303
POT0	$D200
POT1	$D201
POT2	$D202
POT3	$D203
POT4	$D204
POT5	$D205
POT6	$D206
POT7	$D207
POTGO	$D20b
POTSTAT	$D208
POWUPBYT1	$033D
POWUPBYT2	$033E
POWUPBYT3	$033F
PREPLINK	$CA37
PRINTBUF	$03C0
PRMODE	$FF46
PRTBUFPTR	$02DE
PRTBUFSIZ	$02DF
PRWONA	$EF6E
PTIMOUT	$0314
PTRIG0§	$027C
PTRIG1§	$027D
PTRIG2§	$027E
PTRIG3§	$027F
PTRIG4§	$0280
PTRIG5§	$0281
PTRIG6§	$0282
PTRIG7§	$0283
PUTADRESS	$C748
PUTBYTE1	$EF26
PUTCHAR	$C7E3
PUTLINE	$C650
PUTMSC	$F9A6
RAMSIZE	$02E4
RAMTOP	$006A
RAMTSTPTR	$0004
RANDOM	$D20A
READJOY0	$C20E
READJOY1	$C201

DAS INHALTSREGISTER

DAS STEHT DRIN:

ATARI-INTERN ist ein unentbehrliches Arbeitsmittel für jeden, der sich ernsthaft mit Technik und Betriebssystem der ATARI-COMPUTER 600 XL / 800 XL auseinandersetzen will.

Aus dem Inhalt:

- Das Konzept des ATARI
- Die Hardware
- Der ANTIC
  Kontroll- und Lightpenregister
  Bild- und Schriftgrafik
  Scrolling
- Der GTIA
  Darstellung der ANTIC-Bilddaten
  Player-Missile-Grafik
- Der POKEY
  Tonerzeugung
  Tastaturabfrage
  Paddlesteuerung
- Die PIA
- Das Betriebssystem des ATARI
  Reset und Interrupts
- Der Speicherplan

UND GESCHRIEBEN HABEN DIESES BUCH:

Bernd Grohmann ist Student der Elektrotechnik, Lutz Eichler studiert Informatik. Beide haben Erfahrung in der Entwicklung von Hardwareerweiterungen und professioneller Software.

ISBN 3-89011-053-3